CRT-2000-32

Centre de recherche sur les transports (C.R.T.)
Centre for Research on Transportation

GÉNÉRATION AUTOMATIQUE D'HORAIRES D'INFIRMIÈRES À L'AIDE DE LA PROGRAMMATION PAR CONTRAINTES

par

William Feuilletin

TRAVAIL D'ÉTUDIANT

Octobre 2000

101-01816/

PUBLICATION
CRT-2000-32

*GÉNÉRATION AUTOMATIQUE D'HORAIRES D'INFIRMIÈRES À L'AIDE DE LA PROGRAMMATION PAR CONTRAINTES**

par

William Feuilletin[1]

* *Cette recherche a été rendue possible grâce en partie au RCM_2.*

[1] Université Paris Dauphine, Place du Maréchal de Lattre de Tassigny, 75775 Paris cedex 16, France et Centre de recherche sur les transports, Université de Montréal, C.P. 6128, succursale Centre-ville, Montréal, Canada H3C 3J7

Dépôt légal – Bibliothèque nationale du Québec,
Bibliothèque nationale du Canada, 2000

Résumé

Les problèmes d'horaires de ressources humaines constituent depuis longtemps un défi pour ceux qui se sont intéressés à la modélisation mathématique. De nombreuses disciplines mathématiques et informatiques sont abordées pour modéliser et résoudre ces problèmes. L'objectif est d'apporter un outil pratique et performant aux personnes chargées des tâches de confection d'horaires.
Inscrit dans un vaste projet du RCM_2 sur la gestion hospitalière, ce rapport décrit une approche au problème de la génération automatisée d'horaires d'infirmières à l'aide de la programmation par contraintes.
La modélisation mathématique du problème et son implantation à l'aide d'un langage orienté-objet conduisent à des résultats encourageants. Cependant, des améliorations sont à apporter sur les stratégies de recherche de solution.

Mots-clés : modélisation, horaires d'infirmières, programmation par contraintes.

Abstract

For long, human ressources scheduling problems are a challenge for those interested in mathematic modelling. Many disciplines in mathematics and computer science are used to model and solve these problems.
Within a RCM_2 project on optimization in healthcare, this report describes a constraint programming approach to the automated generation of work scheduling for nurses.
A model for this problem and its implementation in an object-oriented language lead to encouraging results. Nevertheless, search strategies should be further investigated.

Key words : modelling, nurses schedules, constraint programming.

Remerciements

Je souhaiterais remercier chaleureusement les personnes qui étaient chargées de l'encadrement du stage.

- Madame Brigitte Jaumard, professeur à l'Ecole Polytechnique, qui m'a donné l'opportunité de travailler sur ce projet. Son support a été essentiel tant sur le plan administratif (permis de travail) que sur le plan scientifique (suivi du projet). Elle est responsable du projet d'opimisation de la gestion hospitalière financé par le RCM_2 et je la remercie pour l'attribution de l'indemnité financière, sésame essentielle pour réaliser un stage à l'étranger.
- Monsieur Gilles Pesant, professeur à l'Ecole Polytechnique, qui m'a enseigné les rudiments de la programmation par contraintes, et dont les conseils ont été précieux pour l'implantation du modèle.
- Monsieur Philippe Galinier, stagiaire post-doctoral à l'Ecole Polytechnique, dont le suivi fut précieux : ses conseils et ses idées ont souvent résolus des problèmes rencontrés lors du projet.

Je voudrais aussi remercier Pascal Labit, étudiant en Maitrise à l'Ecole Polytechnique : ses compétences professionnelles et son omniprésence m'ont permis d'acquérir un grand savoir informatique.
Enfin, je souhaite surtout remercier le corps professoral de l'IUP-GMI de l'Université Paris Dauphine. Cette formation m'a donné un bagage scientifique et m'a surtout permis de réaliser un rêve : passer une année d'études dans un pays étranger. Ce fut pour moi une magnifique opportunité et une expérience inoubliable.

Introduction

La génération d'horaires de ressources humaines est un sujet d'actualité pour les entreprises et les services publics. De nombreux chercheurs ont contribué à des travaux de recherche sur la résolution de ces problèmes d'horaires. L'évolution de la technologie et des découvertes scintifiques, notamment en matière d'informatique et méthodes de recherche opérationnelle, permettent de prendre en compte de manière systématique les contraintes exprimées par les conventions collectives et les préférences du personnel. De plus, l'automatisation de la génération d'horaires permet de libérer les ressources habituellement chargées des tâches de planification.
Mon stage de troisième année d'étude de l'IUP-GMI (Institut Universitaire Professionnel-Génie Mathématique et Informatique) de l'Université Paris-Dauphine (Paris IX), s'est déroulé au C.R.T. (Centre de Recherche sur les Transports), un centre de recherche affilié à quatre institutions universitaires dont l'Ecole Polytechnique de Montréal et dont les bureaux se situent au pavillon des sciences mathématiques et informatiques de l'Université de Montréal. Pendant ce stage, j'ai travaillé sur un projet traitant le problème de la génération automatisée d'horaires pour des unités de soins.
Ce stage a été financé par le R.C.M.$_2$, un consortium regroupant sept centres de recherches dont figurent le C.R.T. et le G.E.R.A.D. (Groupe d'Etude et de Recherche en Analyse de Décisions). Le R.C.M.$_2$ finance plusieurs projet de recherche dont un sur l'optimisation de la gestion hospitalière en collaboration avec le MUHC (McGill University Health Center) et le CHUM informatique (Centre Hospitalier Universitaire de Montréal). Le MUHC est issu du regroupement des hôpitaux anglophones de Montréal dont l'hôpital Royal Victoria avec qui Brigitte Jaumard et son équipe collaborent plus particulièrement. Après un bref aperçu du R.C.M.$_2$ et de deux de ces centres (le G.E.R.A.D. et le C.R.T.) et de leurs domaines de recherche, nous détaillerons le problème de confection automatique d'horaires d'infirmières et l'ensemble de ses contraintes avec une approche mathématique. Par la suite, après avoir présenté la programmation par contraintes, nous verrons la modélisation du problème, et la manière dont il a été implanté à l'aide du logiciel `IlogSolver`. Enfin, nous présenterons quelques résultats et énoncerons quelques hypothèses de travail à développer dans un avenir proche.

Chapitre 1

Le RCM.$_2$, le GERAD et le CRT

1.1 Le R.C.M.$_2$

Crée en 1997, grâce à une subvention de Conseil de recherches en sciences naturelles et en génie (CRSNG), le Réseau de Calcul et de Modélisation Mathématique (R.C.M.$_2$) regroupe sept centres de recherche de la région de Montréal dans une collaboration unique, qui permet de répondre aux besoins de l'industrie dans une grande variété de domaines touchant au calcul et à la modélisation mathématique. Les centres du R.C.M.$_2$ entretiennent des liens étroits avec l'Université de Montréal, l'Université McGill, l'Université du Québec à Montréal (UQAM), l'Université Concordia, l'Université Laval, l'Ecole Polytechnique, l'Ecole des Hautes Etudes Commerciales ainsi que plusieurs universités canadiennes et étrangères. Les centres du RCM_2 sont :

- CERCA : Centre de Recherche en Calcul Appliqué,
- CIRANO : Centre Interuniversitaire de Recherche en Analyse des Organisations,
- CRIM : Centre de Recherche Informatique de Montréal,
- CRM : Centre de Recherches Mathématiques,
- CRT : Centre de Recherche sur les Transports,
- GERAD : Groupe d'Etudes et de Recherche en Analyse des Décisions,
- INRS-Télécommunications : Institut national de la Recherche Scientifique-Télécommunications.

Les personnes qui ont participé à l'encadrement sont des membres de deux de ces centres. B.Jaumard est membre du C.R.T. et du G.E.R.A.D., G.Pesant est membre du C.R.T., et P.Galinier effectue son stage postdoctoral au G.E.R.A.D.
Nous présenterons plus en détail ci-dessous les centres de recherche du G.E.R.A.D. et du C.R.T.

1.2 le GERAD

Le GERAD (Groupe d'études et de recherche en analyse des décisions) est un centre interuniversitaire regroupant des membres de l'Ecole des Hautes Etudes Commerciales, l'Ecole Polytechnique de Montréal, l'Université McGill

et l'Université du Québec à Montréal.
La nécessité de pousser la réflexion sur les fondements de la prise de décision au sein d'organisations privées ou publiques est à l'origine de la création du Groupe d'études et de recherche en analyse des décisions, en 1979. Les activités du GERAD sont orientées vers l'élaboration de systèmes d'aide à la décision. L'approche du groupe se base sur les sciences de la gestion, la recherche opérationnelle et l'analyse des sysèmes. Ses quatre principaux axes de recherche sont les suivants :

- mathématiques de l'analyse de la décision;
- planification de grands systèmes techno-économiques (énergie et communications);
- productique et logistique;
- optimisation des réseaux de communications.

1.3 Le C.R.T.

Le domaine d'expertise du Centre de recherche sur les transports est celui des analyses et méthodes quantitatives et informatiques, appliquées à la planification, à la gestion et à l'exploitation des réseaux de transport urbains, régionaux, interurbains et internationaux, pour les passagers comme pour les marchandises. Il comprend également l'analyse de l'environnement (économique, réglementaire, etc.) et des politiques (gouvernementales, industrielles, etc.) qui les affectent. Le Centre se consacre plus particulièrement à la conception et au développement de méthodes d'évaluation, d'analyse, de simulation et d'optimisation rendues nécessaires par la complexité des problèmes rencontrés en pratique. Son but est de rendre plus efficace et de meilleure qualité le processus d'analyse et de prise de décision qu'impliquent la planification et la gestion des systèmes de transport ainsi que leur réglementation et leur sécurité.
La mission du C.R.T. s'articule autour des trois grands objectifs complémentaires suivants:

- **La recherche-développement**, c'est-à-dire la réalisation de projets et de travaux de recherche théorique et appliquée, ainsi que d'activités de développement et de démonstration dans son domaine de compétence, englobant toutes les disciplines scientifiques et les techniques pertinentes à ce domaine.
- **La formation** de spécialistes en transport et dans les disciplines concernées, par la direction et l'encadrement d'étudiants de 2e et 3e cycles (et, à l'occasion, 1er cycle), ainsi que de chercheurs postdoctoraux et d'agents de recherche.
- **Le transfert scientifique et technologique Université → Industrie / Organismes publics**, c'est-à-dire le transfert de connaissances et de techniques spécialisées auprès des entreprises, agences gouvernementales et autres organismes oeuvrant dans le domaine du transport ou reliés par certaines activités à ce domaine.

Il est opportun de souligner ici la place de premier choix qu'occupe le C.R.T. dans la planification d'ensemble de l'Université de Montréal et de ses écoles affiliées. Depuis plus de vingt ans, celles-ci ont pu constater que, dans un domaine des plus importants au niveau socio-économique et industriel, le C.R.T. a su démontrer de façon évidente sa capacité à participer à la mission de l'Université sur le plan de la recherche, de la formation et du transfert scientifique et technologique. En fait, le rôle essentiel du C.R.T. a été défini dès sa création en 1971, lorsque l'Université de Montréal a reconnu l'importance de développer la recherche et la formation pluridisciplinaire dans le domaine des transports. Celui-ci constitue d'ailleurs un domaine d'application privilégié

des disciplines comprises dans l'axe de développement Mathématiques-Informatique-Recherche opérationnelle, reconnu en 1976 par le Conseil des Universités du Québec à l'Université de Montréal et accepté par celle-ci. La mme constatation est tout ai si vraie en ce qui concerne d'autres disciplines majeures de l'Université de Montréal et de ses écoles affiliées, telles l'économie, la médecine ou le génie.

1.4 La programmation scientifique du C.R.T.

1.4.1 Les objectifs

Les objectifs de recherche du C.R.T. demeurent essentiellement les mêmes. Ils visent, d'abord et avant tout, l'avancement des connaissances et des technologies en transport, conjugués à la volonté de confronter les produits de sa recherche à des situations réelles. Ces objectifs se traduisent par la réalisation de projets de recherche et développement, fondamentaux et appliqués, pertinents et de qualité, ainsi que par la formation de chercheurs et de spécialistes de haut niveau susceptibles de s'intégrer ensuite dans diverses institutions, organismes et entreprises. Ils se traduisent aussi par l'opportunité offerte à ces entreprises d'acquérir de nouvelles approches et méthodes de travail, ainsi que de nouveaux instruments qui leur permettront de faciliter la formulation de leurs problèmes, de rationaliser leurs stratégies ou encore de rendre leurs opérations plus efficaces et plus productives.
La programmation scientifique du C.R.T. représente l'expression détaillée de ces objectifs. Elle regroupe, sous différents axes de recherche, tous les projets qui seront réalisés ou entrepris au cours des prochaines années. Selon la phase dans laquelle ils se trouvent, ces projets seront orientés vers la recherche fondamentale, la recherche appliquée ou encore vers le développement de "prototypes". Toutefois, ils visent tous à rendre plus efficace et de meilleure qualité le processus d'analyse et de prise de décisions dans le domaine de la planification et de l'exploitation des réseaux de transport.

1.4.2 Axes et programmes de recherche

En tenant compte des voies stratégiques établies et des objectifs de recherche, la programmation scientifique 1996-2000 s'articule autour de sept axes de recherche.

Planification des systèmes de transport. Cet axe couvre les projets de recherche visant à améliorer la qualité, l'efficacité, la facilité d'utilisation et la validité de différentes techniques d'analyse et de planification des réseaux de transport de personnes et de marchandises en milieu urbain, régional ou international.

Exploitation des réseaux de transport. Cet axe porte sur le développement de techniques d'élaboration de tournées et d'horaires et de techniques de répartition appliquées à la gestion des réseaux exploités par des compagnies de transport de passagers ou de marchandises.

Systèmes intelligents de transport. Cet axe de recherche touche l'analyse et le développement de systèmes en temps réel de gestion de la circulation, de guidage routier, de prévision du trafic et du temps de déplacement, de gestion pour flottes de véhicules de passagers et de marchandises. Il couvre également le développement de bibliothèques de logiciels pouvant tre utilisés sur des plates- formes de calcul parallèle, ainsi que des simulateurs de trafic.

Economie et politique des transports. Cet axe touche les problèmes ayant trait aux coûts, à la demande, au marché, aux tarifs, à la productivité, à la performance et à la réglementation en matière de transport. Il couvre l'étude des problèmes méthodologiques, l'étude du comportement des marchés, la gestion des risques ainsi que l'étude de la demande de transport et du cadre institutionnel déterminant l'offre de transport.

Sécurité dans les transports. Cet axe de recherche touche l'analyse des phénomènes humains, socio-économiques et physiques en matière de transport, ainsi que l'évaluation et la détermination des conditions ou des mesures susceptibles de réduire le nombre d'accidents, leur gravité et leur coût.

Planification de l'entretien des infrastructures. Cet axe vise la conception et le développement de systèmes informatisés d'aide à l'analyse et à la prise de décision dans le domaine de la planification et de la gestion des interventions de réhabilitation et d'entretien des réseaux routiers et autres infrastructures.

Télécommunications. Cet axe de recherche porte sur le développement de modèles et d'algorithmes pour résoudre les problèmes d'acheminement, de dimensionnement et de synthèse de réseau dans les réseaux de télécommunications .

Chapitre 2

Introduction au problème

2.1 Introduction

Ce rapport décrit une approche par la programmation par contraintes, du problème de la génération automatique des horaires d'infirmières : ce n'est qu'un exemple particulier de l'ensemble des problèmes liés aux horaires des ressources humaines.
Nous appliquerons notre étude sur des unités de soins disposant d'une cinquantaine d'infirmières. Même si ce problème semble relativement de petite taille, le défi consiste à générer des horaires équilibrés entre les différentes infirmières. Les difficultés de l'exercice se révèlent à plusieurs niveaux : il apparait évident que toutes les conditions de travail, définies par les conventions sociales, devront être respectées. Cela concerne non seulement les conditions individuelles (une infirmière doit disposer d'un horaire conforme aux règles régies par les conventions et approprié à ses aptitudes), mais aussi les conditions générales (l'hôpital doit garantir les conditions minimales de sécurité et de qualité pour une bonne pratique des soins). Toutes ces conditions de travail apparaissent ici comme des contraintes que nous devons satisfaire afin de générer des horaires de qualité.
Ce problème a déjà été traité de différentes manières par les étudiants de Madame Jaumard :

- Pascal Labit et Tsévi Vovor ont utilisé la méthode de la génération de colonnes dans le cadre d'un contrat avec l'hôpital Royal Victoria et le CHUM (Centre Hospitalier de l'Université de Montréal). Ceci a conduit au logiciel IRIS sur le point d'être implanté dans diverses unités de soins de l'hôpital Royal Victoria.
- Philippe Galinier a utilisé la méthode de recherche (méthode tabou) pour développer un autre programme HORINF de confection d'horaires d'infirmières.

Nous allons étudier le problème lié à la génération d'horaires d'infirmières, en nous basant sur des jeux de données fournis par une unité de soins de l'hôpital Royal Victoria de Montréal : ces données sont constituées d'informations générales (données numériques propres au problème traité, listes de contraintes de groupe) et personnelles (données relatives pour chaque infirmière, liste de contraintes individuelles).

2.2 Description du problème

Soit $\mathcal{N}$, l'ensemble des infirmières (nurses), de cardinal N. A chaque infirmière, est attribué un niveau d'expertise (level) $l \in \mathcal{L}$, l'ensemble des niveaux d'expertise, de cardinal L. Le problème consiste à définir, pour chaque

infirmière, un horaire (ou emploi du temps) sur l'ensemble $\mathcal{D}$ des jours (days) de l'horizon (ou planning), de cardinal D. Le premier et le dernier jour sont fournis par les données.
Soit $\mathcal{S}$, l'ensemble des quarts (shifts) de travail, de cardinal S (par exemple, L'hôpital Royal Victoria utilise les quarts suivants {D,D12,E,E1,E2,N,S,S12}). L'ensemble des quarts couvrent la totalité de la journée de travail. A chaque quart, sont associés le nombre d'heures payées, ainsi que le type de quart (day, evening, night) : par exemple, D représente un quart de jour d'une durée de 8 heures, S12 représente un quart de nuit d'une durée de 12 heures,...etc (voir Annexes). A ces quarts-là, nous rajoutons un quart "Off" correspondant à un jour de repos, et un quart "Vacation" correspondant à un jour de vacances (congés payés).
Parce que les quarts peuvent se "recouper" (l'intersection de deux quarts n'étant pas nécéssairement vide), nous faisons appel à une autre subdivision de la journée : les périodes. Elles forment une partition de la durée de la journée. Il s'agit de la partition la plus grossière telle que tout quart s'exprime comme une réunion de périodes consécutives. Une période sera donc couverte par une liste de quarts, tout comme un quart pourra couvrir une suite de périodes consécutives.
De nombreuses contraintes interviennent dans la conception des horaires : les principales contraintes portent sur les quotas (nombre d'infirmières présentes à une période donnée), la charge de travail d'une infirmière, l'attribution de ses jours de congé, le nombre de quarts consécutifs d'un même type,...etc.
Parmi les horaires réalisables, tous ne sont pas également souhaitables ; on utilise une fonction d'évaluation pour quantifier la qualité de chacun des horaires. Le problème consiste à concevoir un horaire réalisable qui maximise la fonction d'évaluation. Les données du problème, les contraintes et la fonction d'évaluation sont décrits dans les sections suivantes.

Le problème peut se définir ainsi : étant donnés un ensemble d'infirmières ayant chacune un niveau d'expertise, un ensemble de jours sur un horizon, et un ensemble de quarts de travail et de repos, ainsi qu'un ensemble de règles pour la construction des horaires (règles individuelles et générales), affecter un quart pour chaque infirmière et chaque jour de l'horizon. Pour toute infirmière $n \in \mathcal{N}$, et pour tout jour $d \in \mathcal{D}$, nous définissons une variable $\sigma_{n,d}$, qui prend ses valeurs dans $\mathcal{S}$.

$$\begin{array}{ccc} \mathcal{N} \times \mathcal{D} & \longrightarrow & \mathcal{S} \\ (n,d) & \mapsto & \sigma_{n,d} \end{array}$$

Une solution du problème est dite réalisable lorsque pour tout couple $(n,d) \in \mathcal{N} \times \mathcal{D}$, il existe une valeur qui satisfait toutes les contraintes (générales et individuelles).

exemple Une solution du problème sera donnée sous la forme d'un tableau, représentant l'ensemble des emplois du temps de chaque infirmière sur l'horizon.

	Mon	*Tue*	*Wed*	*Thu*	*Fri*	*Sat*	*Sun*
$Nurse_1$	*D*	*D*	*D*	*D*	*Off*	*Off*	*S12*
$Nurse_2$	*S*	*S*	*Off*	*Off*	*D12*	*D12*	*E1*
⋮							
$Nurse_n$	*Off*	*Off*	*D*	*D*	*E*	*E*	*E2*

2.3 Présentation des contraintes

A ce problème, sont associées un ensemble de contraintes individuelles et générales. Cependant, nous discernons deux types de contraintes. Les contraintes dures (hard constraints) sont des contraintes qui sont

systématiquement satisfaites, quelle que soit la solution finale proposée. Les contraintes molles (soft constraints) ne sont pas systématiquement satisfaites, mais servent d'indicateur sur la qualité de la solution proposée. En effet, nous utilisons une fonction d'évaluation (ou fonction coût) pour quantifier la qualité des horaires. Dans le problème que nous traitons, nous ne parlerons que de contraintes et de fonction objectif.

2.3.1 Les contraintes dures (hard constraints)

Les pré-affectations (preassignments) Motivée par des raisons personnelles ou professionnelles, une infirmière peut se voir attribuer d'office des quarts de travail sur l'horizon. Il s'agit de quarts qui seront systématiquement accordés à l'infirmière considérée, quelle que soit la solution proposée. Dans les données fournies par l'hôpital, une liste $\mathcal{PA}_n$ de pré-affectations est définie pour chaque infirmière $n \in \mathcal{N}$. Chaque pré-affectation p de la liste $\mathcal{PA}_n$ est définie par un couple $(d_{n,p}, s_{n,p})$, où

- $d_{n,p} \in \mathcal{D}$ est le jour de la p^{eme} pré-affectation de la n^{eme} infirmière.
- $s_{n,p} \in \mathcal{S}$ est le quart de la p^{eme} pré-affectation de la n^{eme} infirmière.

La contrainte s'exprime ainsi : pour toutes les solutions proposées, pour tout $n \in \mathcal{N}$, pour tout $p \in \mathcal{PA}_n$, le quart travaillé par l'infirmière n le jour $d_{n,p}$ est le quart $s_{n,p}$.

Formulation mathématique de la contrainte

$$\forall n \in \mathcal{N}, \forall p \in \mathcal{PA}_n, \quad \sigma_{n,d_{n,p}} = s_{n,p}$$

Les affectations interdites (forbidden assignments) Au même titre que pour les pré-affectations, une infirmière peut se voir refuser l'attribution de certains quarts de travail sur certains jours de l'horizon. Ces quarts ne seront jamais accordés à l'infirmière considérée, quelque soit la solution proposée. Une liste $\mathcal{FA}_n$ est définie pour chaque infirmière $n \in \mathcal{N}$. Chaque affectation interdite p de la liste $\mathcal{FA}_n$ est définie par un couple $(d_{n,p}, s_{n,p})$, où :

- $d_{n,p} \in \mathcal{D}$ est le jour de la p^{eme} pré-affectation de la n^{eme} infirmière.
- $s_{n,p} \in \mathcal{S}$ est le quart de la p^{eme} pré-affectation de la n^{eme} infirmière.

La contrainte s'exprime ainsi : pour toutes les solutions proposées, pour tout $n \in \mathcal{N}$, pour tout $p \in \mathcal{FA}_n$, le quart travaillé par l'infirmière n le jour $d_{n,p}$ est différent du quart $s_{n,p}$.

Formulation mathématique de la contrainte

$$\forall n \in \mathcal{N}, \forall p \in \mathcal{FA}_n, \quad \sigma_{n,d_{n,p}} \neq s_{n,p}$$

Les quarts réalisables (feasible shifts) D'une manière générale, toute infirmière travaillant pour un service quelconque, est susceptible de se voir affecter n'importe quel quart de travail. Cependant, dans la réalité, les conventions de travail peuvent restreindre l'ensemble des quarts affectés à certains types d'infirmières. Par exemple, une infirmière de faible niveau de qualification peut se voir refuser les affectations de tous les quarts de nuit.
Une liste $\mathcal{FS}_n$ est définie pour chaque infirmière $n \in \mathcal{N}$. Cette liste est un sous-ensemble de $\mathcal{S}$, et contient les quarts réalisables pour l'infirmière considérée. La contrainte peut s'exprimer ainsi : pour toutes les solutions réalisables, pour tout $n \in \mathcal{N}$, pour tout $d \in \mathcal{D}$, le quart travaillé par l'infirmière n, le jour d, appartient à l'ensemble des quarts réalisables, défini par la liste $\mathcal{FS}_n$.

Formulation mathématique de la contrainte

$$\forall n \in \mathcal{N}, \forall d \in \mathcal{D}, \quad \sigma_{n,d} \in \mathcal{FS}_n$$

Les quarts du lendemain (next day shifts) Afin de respecter une certaine logique dans l'enchainement des quarts sur l'horizon d'une infirmière, nous imposons certaines règles concernant les relations entre les quarts sur deux jours consécutifs. Par exemple, nous pouvons interdire à une infirmière de travailler le quart de nuit d'un certain jour, et le quart du matin du jour suivant.
Pour cela, nous définissons, pour chaque quart de travail s de l'ensemble $\mathcal{S}$, une liste $\mathcal{ND}_s$ de quarts réalisables le jour suivant. Si, pour tout infirmière n de $\mathcal{N}$, et pour tout jour d de $\mathcal{D}$-{dernier jour de l'horizon}, la variable $\sigma_{n,d}$ est fixée à la valeur s, alors la variable $\sigma_{n,d+1}$ prend ses valeurs parmi celles de la liste $\mathcal{ND}_s$.

Formulation mathématique $\forall n \in \mathcal{N}, \forall d \in \mathcal{D} - \{d_{max} = \text{last day}\}, \forall s \in \mathcal{S},$

$$\sigma_{n,d} = s \quad \Rightarrow \quad \sigma_{n,d+1} \in \mathcal{ND}_s$$

Les charges de travail (workload) Les charges de travail sont définies à l'aide des conventions sociales. Elles sont calculées en heures et en minutes, sur un certain nombre de jours, et varient en fonction des infirmières (niveau d'expertise, expérience,...). Lorsqu'une infirmière travaille un quart, sa charge de travail est incrémentée du nombre d'heures payées relatif au quart travaillé (`hoursPaid(shifts)`). Pour chaque infirmière $n \in \mathcal{N}$, nous définissons une liste $\mathcal{W}_n$ de contraintes w portant sur les charges de travail. Chaque contrainte w est définie comme suit :

- $startingDay_{n,w}$, une date de début.
- $endDay_{n,w}$, une date de fin.
- $minHoursPaid_{n,w}$, un nombre minimum d'heures payées.
- $maxHoursPaid_{n,w}$, un nombre maximum d'heures payées

La contrainte peut s'exprimer ainsi : Pour toute infirmière $n \in \mathcal{N}$, pour toute contrainte w de la liste $\mathcal{W}_n$, la somme des heures payées en fonction des quarts travaillés par l'infirmière n pendant la période délimitée par la date de début $startingDay_{n,w}$ et la date de fin $endDay_{n,w}$, doit être comprise entre le minimum et le maximum d'heures payées ($minHoursPaid_{n,w}$ et $maxHoursPaid_{n,w}$).

Formulation mathématique $\forall n \in \mathcal{N}, \forall w \in \mathcal{W}_n,$

$$minHoursPaid_{n,w} \leq \sum_{d=startingDay_{n,w}}^{endDay_{n,w}} hoursPaid(\sigma_{n,d}) \leq maxHoursPaid_{n,w}$$

Les quotas Afin d'assurer le bon fonctionnement des services hospitaliers en matière de qualité des soins, et de disponibilité des infirmières, il est nécessaire d'imposer des règles de quotas, c'est-à-dire, un nombre obligatoire d'infirmières à un moment donné de la journée.
Le personnel hospitalier du Royal Victoria travaille selon des quarts pré-définis : ils sont de taille variable (nombre d'heures différent), et répartis de manière à couvrir l'ensemble des 24 heures de la journée. De plus, certaines plages horaires peuvent être couvertes par plusieurs quarts si ceux-ci se chevauchent. Afin de mieux

gérer la quantification des quotas, nous réalisons une subdivision de la journée en périodes : elle tient compte des recouvrements des quarts de travail. Si deux quarts ne se recouvrent pas (par exemple : matin de 6h à 14h, après-midi de 14h à 22h), alors il y aura deux périodes correspondant aux deux quarts. Dans le cas contraire (matin de 6h à 14h, après-midi de 12h à 20h), il y aura trois périodes dont l'une va couvrir l'intervalle de temps durant lequel les deux quarts se recouvrent (voir figure 2.1).

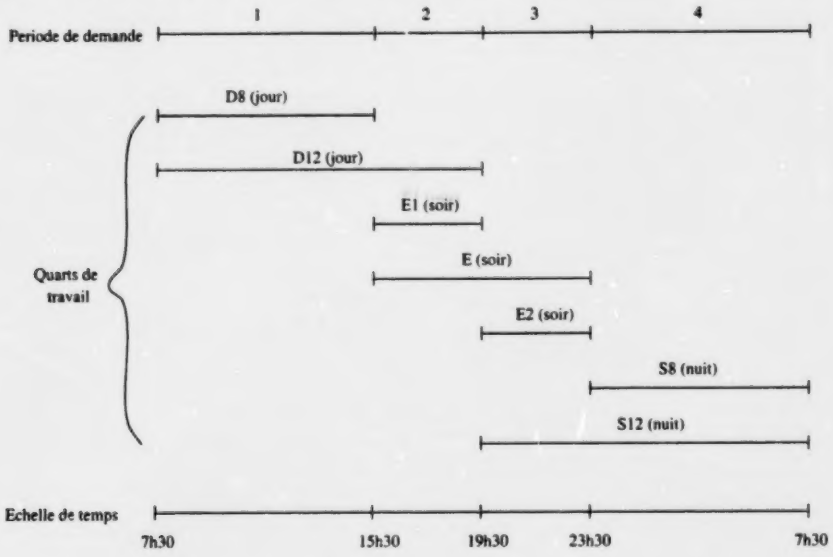

Figure 2.1: Les périodes : une partition de la journée

Soit $\mathcal{Q}$, la liste des contraintes de quotas : elles vont alors porter sur les périodes de la journée. A chaque contrainte $q \in \mathcal{Q}$, nous associons :

- Une liste de niveaux ($\mathcal{L}_q$): elle correspond à l'ensemble des niveaux d'expertise que les infirmières doivent posséder pour être habilitées à travailler durant cette période.
- Une liste de quarts ($\mathcal{S}_q$) : elle correspond à l'ensemble des quarts qui couvrent la période considérée.
- Le sens de la contrainte (sense_q) : c'est un caractère choisi parmi ($=, \neq, \leq$), qui définit le choix selon lequel la contrainte considérée est une contrainte d'égalité ou d'inégalité.
- Un entier quota_q : il définit le quota-cible de la contrainte, celui que nous chercherons à atteindre pour optimiser la solution.
- Un entier max_surplus_q : il définit la différence entre le quota maximum autorisé, et le quota-cible.
- Un entier max_deficit_q : il définit la différence entre le quota-cible et le quota minimum autorisé.
- Une période (period_q): elle correspond à la période pour laquelle la contrainte est considérée.
- Un jour (day_q): il correspond au jour pour lequel la contrainte est considérée .

La contrainte peut alors s'exprimer ainsi : Pour toute contrainte q de l'ensemble des contraintes de quotas, soit $\mathcal{N}urse_q$, l'ensemble des infirmières dont level(n) (le niveau d'expertise de l'infirmière n) appartient à $\mathcal{L}_q$, et travaillant dans un des quarts de $\mathcal{S}_q$ pendant la journée day_q. Alors le cardinal de l'ensemble $\mathcal{N}urse_q$ appartient à l'intervalle borné par $\text{quota}_q - \text{max_deficit}_q$ et $\text{quota}_q + \text{max_surplus}_q$.

Formulation mathématique Soit :

$$\mathcal{N}urse_q = \{n \in \mathcal{N} | Level(n) \in \mathcal{L}_q \wedge \sigma_{n,day_q} \in \mathcal{S}_q\}$$

Alors,

$$quota_q - max_deficit_q \leq |\mathcal{N}urse_q| \leq quota_q + max_surplus_q$$

Les congés (vacations) Dans l'horizon, une infirmière peut se voir offrir un certain nombre de jours de congé. Ce nombre est borné inférieurement et supérieurement sur un intervalle de temps donné. Ces jours sont à prendre parmi une liste de jours candidats. Pour chaque infirmière $n \in \mathcal{N}$, nous définissons :

- un paramètre d'activation de la contrainte ($\text{active_rule_vacations}_n$) : il détermine, par sa valeur (0/1), si la contrainte est à prendre en compte pour l'infirmière n.
- une date de début de période (startingDay_n).
- une date de fin de période (endingDay_n).
- un nombre minimal de jours de congés à accorder ($\text{vacations_min_days}_n$).
- un nombre maximal de jours de congés à accorder ($\text{vacations_max_days}_n$).
- une liste de jours candidats ($\mathcal{V}_n$).

La contrainte peut s'exprimer ainsi : Pour toute infirmière $n \in \mathcal{N}$, si le paramètre d'activation de la contrainte $\text{active_rule_vacations}_n$ est égal à 1, alors le nombre de jours de congés accordés parmi la liste $\mathcal{V}_n$ de jours candidats (ces jours étant compris entre la date de début startingDay_n et la date de fin endingDay_n), est compris entre les nombres $\text{vacations_min_days}_n$ et $\text{vacations_max_days}_n$, les nombres minimal et maximal de jours de congés à accorder.De même, pour tous les autres jours de l'horizon n'appartenant pas à $\mathcal{V}_n$, aucun jour de congé n'est accordé à l'infirmière n.

Remarque : si le paramètre d'activation $\text{active_rule_vacations}_n$ est égal à zéro, alors aucun jour de congé n'est accordé à l'infirmière n durant l'horizon courant.

Formulation mathématique $\forall n \in \mathcal{N}$, if $active_rule_vacations_n = 0$, then

$$\forall d \in \mathcal{D}, \sigma_{n,d} \neq Vacation$$

else

$$vacations_min_days_n \leq |\{d \in \mathcal{V}_n \wedge \sigma_{n,d} = Vacation\}| \leq vacations_max_days_n$$
$$\forall d \in \mathcal{D} \backslash \mathcal{V}_n, \sigma_{n,d} \neq Vacation$$

Les quarts consécutifs (consecutive shifts) Durant une suite de jours consécutifs pour laquelle une infirmière est de service, il arrive qu'elle puisse travailler pendant plusieurs quarts du même type (day, evening, night) sur plusieurs jours d'affilée. Afin d'équilibrer la répartition des types de quarts entre les différentes infirmières, nous instaurons, pour chaque infirmière, des bornes minimale et maximale pour chaque type de quart (cela concerne aussi les quarts de type "off").
Par exemple, supposons que pour une infirmière donnée, le nombre de quarts consécutifs de jour et de nuit soit compris entre 2 et 3. Alors la figure 2.2 représente un cas non réalisable, alors que la figure 2.3 représente un cas réalisable.

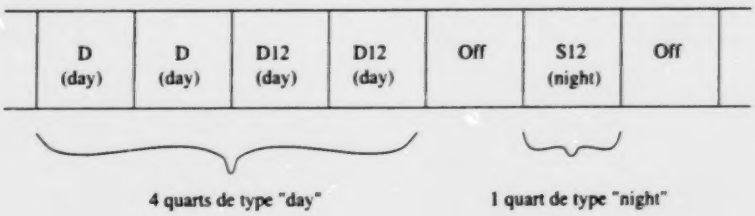

Figure 2.2: types de quarts consécutifs : cas non réalisable

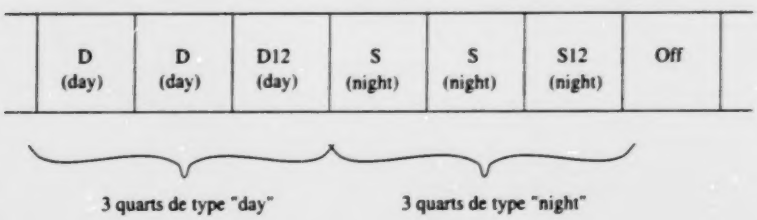

Figure 2.3: types de quarts consécutifs : cas réalisable

La contrainte peut s'exprimer ainsi : Pout toute infirmière $n \in \mathcal{N}$, le nombre de quarts consécutifs d'un même type $t \in \{day, evening, night\}$, est compris entre les bornes minimales et maximales définies pour chaque infirmière et pour chaque type de quart.

Formulation mathématique Pour toute infirmière $n \in \mathcal{N}$, et pour chaque type $t \in \{day, evening, night\}$, nous définissons les bornes minimales et maximales :

- $min_{n,t}$ est le minimum de quarts consécutifs de type t réalisable par l'infirmière n.
- $max_{n,t}$ est le maximum de quarts consécutifs de type t réalisable par l'infirmière n.

Alors, $\forall n \in \mathcal{N}$, $\forall d \in \mathcal{D}$,

$$\{i | type(\sigma n, d+j) = t \ \ \forall j \in [0, 1, \ldots, i] \wedge type(\sigma_{n,d-1}) \neq t\} \in [min_{n,t}, max_{n,t}]$$

Les fins de semaines consécutives (consecutive week-ends) Nous considérons qu'une infirmière travaille une fin de semaine lorqu'elle travaille au moins un quart soit dans la journée de samedi, soit dans la journée de dimanche. Il existe donc plusieurs scénari pour savoir si une infirmière travaille une fin de semaine ou non.

- un jour est non travaillé si le quart affecté pour ce jour est le quart "Off".
- un jour est un congé si le quart affecté pour ce jour est le quart "Vacation".
- un jour est travaillé si le quart affecté pour ce jour est un quart de l'ensemble $\mathcal{S}/\{Off, Vacation\}$

scénario	samedi	dimanche	week-end
1	travaillé	travaillé	travaillé
2	travaillé	non travaillé	travaillé
3	travaillé	congé	travaillé
4	non travaillé	travaillé	travaillé
5	non travaillé	non travaillé	non travaillé
6	non travaillé	congé	non travaillé
7	congé	travaillé	travaillé
8	congé	non travaillé	non travaillé
9	congé	congé	non travaillé

De la même manière que pour les quarts consécutifs, les nombres de fins de semaine consécutives travaillées et non travaillées par une infirmière n sont bornés inférieurement et supérieurement.

2.3.2 La fonction objectif (cost function)

Les fins de semaines brisées (broken week-ends) Dans la contrainte dure sur les fins de semaine consécutives, nous avons vu que des fins de semaine travaillées pouvaient être composées d'un jour travaillé et d'un jour non travaillé. Ce type de fin de semaine est appelé fin de semaine brisée ($brokenWeekEnd$). Nous souhaitons éviter ce genre de scénario, afin que les infirmières puissent profiter de l'intégralité de leur fin de semaine.

scénario	samedi	dimanche	week-end
1	travaillé	non travaillé	brisé
2	non travaillé	travaillé	brisé
3	travaillé	congé	brisé
4	congé	travaillé	brisé

Soit $\mathcal{WE}$, la liste des fins de semaine, définie par :

$$\mathcal{WE} = \{(d, d+1) | d \in \mathcal{D}, d \text{ est un samedi}\}$$

Nous définissons par $aversion_for_no_long_weekend$ (A), un coefficient qui pénalise la fonction objectif lorsqu'une fin de semaine est brisée. Ce coefficient est un rationnel compris entre -100 et 0.
La pénalité s'exprime ainsi : Pour toute infirmière $n \in \mathcal{N}$, pour toute fin de semaine $w \in \mathcal{WE}$, le coefficient $aversion_for_no_long_weekend$ est incrémenté à la fonction objectif si la fin de semaine w de l'infirmière n est brisée.

Formulation mathématique Soit $isBrokenWeek\mathcal{E}nd$, une fonction définie de la manière suivante :

$$\begin{array}{rcl} \mathcal{N} \times \mathcal{WE} & \longrightarrow & [0;1] \\ (n,w) & \longmapsto & \text{1 si la fin de semaine } w \text{ de l'infirmière } n \\ & & \quad \text{est brisée.} \\ & & \text{0 sinon.} \end{array}$$

Soit $Cost_{\mathcal{WE}}$, la contribution relative au coût des fins de semaine brisées

$$Cost_{\mathcal{WE}} = \sum_{n \in \mathcal{N}} \sum_{w \in \mathcal{WE}} A * isBrokenWeek\mathcal{E}nd(n,w)$$

Les quotas Nous avons vu, dans les contraintes dures, que l'hôpital imposait des quotas sur la présence des infirmières pour une période donnée. Les valeurs cibles des demandes de quotas peuvent être considérées comme des contraintes molles, pouvant être satisfaites avec un surplus ou un déficit spécifié. Des pénalités sont associées aux déviations par rapport aux valeurs cibles, et interviennent dans la fonction objectif : ces pénalités sont des valeurs négatives et sont mentionnées dans les données par *Aversion_For_Overstaffing* (A^+) et *Aversion_For_Understaffing* (A^-). Pour toute contrainte de quotas $q \in \mathcal{Q}$, soient

- $\mathcal{N}_q$, le nombre d'infirmières travaillant le jour day_q pendant la période $period_q$.
- $quota_q$, la valeur-cible.

remarque : day_q, $period_q$ et $quota_q$ sont définis dans la contrainte q).
La contrainte peut s'exprimer ainsi : pour toute contrainte $q \in \mathcal{Q}$, toute dépassement inférieur (resp. supérieur) de la valeur cible $quota_q$, par $\mathcal{N}_q$, est sanctionné par une pénalité A^- (resp. A^+).

Formulation mathématique Soit $Cost_{\mathcal{Q}}$, la contribution relative au coût des quotas.

$$Cost_{\mathcal{Q}} = \sum_{q \in \mathcal{Q}} A^+ * \max(0, \mathcal{N}_q - quota_q) - A^- * \min(0, \mathcal{N}_q - quota_q)$$

Les préférences (aversions) individuelles (individual preferencesand aversions) Pour des raisons professionnelles et personnelles, une infirmière peut vouloir travailler un jour précis de l'horizon, tout comme elle peut souhaiter être dans les équipes de nuit pendant une semaine donnée. Au contraire, elle peut souhaiter ne pas être affectée à tel quart pour un jour donné. Ces considérations seront prises en compte lors de l'élaboration de l'emploi du temps, mais ne sont pas systématiquement satisfaites.
Pour tout infirmière $n \in \mathcal{N}$, nous définissons une liste $\mathcal{P}r_n$. Chaque préférence p de $\mathcal{P}r_n$ est définie par :

- $d_{n,p} \in \mathcal{D}$, le jour de la p^{eme} préférence de la n^{eme} infirmière.
- $s_{n,p} \in \mathcal{S}$, le quart de la p^{eme} préférence de la n^{eme} infirmière.
- $value_{n,p} \in [-10;10]$, un coefficient rationnel de pondération, correspondant à la contribution de la p^{eme} préférence de la n^{eme} infirmière.

Remarque : si la valeur de *value* est positive, cela correspond à une préférence de l'infirmière. Dans le cas contraire, nous aurons affaire à une aversion de la part de l'infirmière.
La pénalité peut s'exprimer de la manière suivante : Pour toute infirmière $n \in \mathcal{N}$, pour toute préférence p de la liste $\mathcal{P}r_n$, si l'infirmière n est affectée au quart $s_{n,p}$ le jour $d_{n,p}$, alors la valeur $value_{n,p}$ est incrémentée à la fonction objectif $Cost$.

Formulation mathématique Nous définissons par $isSatisfied$, la fonction telle que pour tout $n \in \mathcal{N}$, pour tout $p \in \mathcal{P}r_n$,

$$\begin{aligned} isSatisfied(n,p) &= 1 \text{ si } \sigma_{n,d_p} = s_p. \\ &= 0 \text{ sinon.} \end{aligned}$$

Soit $Cost_{\mathcal{P}r}$, la contribution relative au coût des préférences et aversions individuelles

$$Cost_{\mathcal{P}r} = \sum_{n \in \mathcal{N}} \sum_{p \in \mathcal{P}r_n} isSatisfied(n,p) * value_{n,p}$$

Les ratios des quarts (shift ratios) Afin de respecter une certain équilibre entre les différents quarts effectués par les infirmières, nous instaurons des ratios sur les types de quarts (day, evening, night). Le ratio d'un type exprime le pourcentage de quarts travaillés appartenant à ce type, parmi l'ensemble des quarts travaillés par une infirmière. A chaque type de quart, sont associés deux ratios, l'un étant une borne inférieure, l'autre supérieure. Notons cependant que les ratios sont définis individuellement pour chaque infirmière : en effet, si une infirmière souhaite travailler plus de quarts de nuit que de quarts de jour, nous pouvons augmenter les ratios des quarts de nuit au détriment des ratios de quarts de jour. Ainsi, pour chaque infirmière $n \in \mathcal{N}$, nous définissons :

- $maxDayShiftRatio_n$, le pourcentage maximum de quarts de jour pour l'infirmière n
- $minDayShiftRatio_n$, le pourcentage minimum de quarts de jour pour l'infirmière n
- $maxEveningShiftRatio_n$, le pourcentage maximum de quarts de soir pour l'infirmière n
- $minEveningShiftRatio_n$, le pourcentage minimum de quarts de soir pour l'infirmière n
- $maxNightShiftRatio_n$, le pourcentage maximum de quarts de nuit pour l'infirmière n
- $minNightShiftRatio_n$, le pourcentage minimum de quarts de nuit pour l'infirmière n

Nous exprimerons la pénalité de la manière suivante : pour tout infirmière $n \in \mathcal{N}$, pour chaque type t de quart, le pourcentage des quarts de type t travaillés par l'infirmière n pendant l'horizon, est borné par les pourcentages minimal et maximal des quarts de type t définis pour l'infirmière n. Une pénalité est attribuée pour tout dépassement inférieur (resp. supérieur) de la borne minimale (resp. maximale).

Formulation mathématique Soit $\mathcal{WS}_n$, la liste des quarts travaillés par l'infirmière $n \in \mathcal{N}$ pendant l'horizon. Pour tout $n \in \mathcal{N}$, Nous posons

$$\begin{aligned} ratio_day_n &= \frac{|\{d \in \mathcal{D} : type(\sigma_{n,d}) = day\}|}{|\{d \in \mathcal{D} : \sigma_{n,d} \in \mathcal{WS}_n\}|} \\ ratio_evening_n &= \frac{|\{d \in \mathcal{D} : type(\sigma_{n,d}) = evening\}|}{|\{d \in \mathcal{D} : \sigma_{n,d} \in \mathcal{WS}_n\}|} \\ ratio_night_n &= \frac{|\{d \in \mathcal{D} : type(\sigma_{n,d}) = night\}|}{|\{d \in \mathcal{D} : \sigma_{n,d} \in \mathcal{WS}_n\}|} \end{aligned}$$

Soient A, la pénalité attribuée en cas de dépassement, et $Cost_{ratio}$, la contribution relative au coût des ratios.

$$\begin{aligned} Cost_{ratio} &= A * \sum_{n \in \mathcal{N}} \max(0, minDayShiftRatio_n - ratio_day_n) \\ &+ A * \sum_{n \in \mathcal{N}} \max(0, minEveningShiftRatio_n - ratio_evening_n) \\ &+ A * \sum_{n \in \mathcal{N}} \max(0, minNightShiftRatio_n - ratio_night_n) \\ &+ A * \sum_{n \in \mathcal{N}} \max(0, ratio_day_n - maxDayShiftRatio_n) \\ &+ A * \sum_{n \in \mathcal{N}} \max(0, ratio_evening_n - maxEveningShiftRatio_n) \\ &+ A * \sum_{n \in \mathcal{N}} \max(0, ratio_night_n - maxNightShiftRatio_n) \end{aligned}$$

2.3.3 La fonction objectif

Soit$Cost$, la fonction objectif, liée au coût global de la solution. $Cost$ est la somme de toutes les contributions relative aux contraintes molles.

$$Cost = Cost_{\mathcal{WE}} + Cost_{\mathcal{Q}} - Cost_{\mathcal{P}r} + Cost_{ratio}.$$

Nous minimiserons la fonction objectif, afin d'obtenir la meilleure solution possible.

Chapitre 3

La programmation par contraintes

Avec la programmation par contraintes, un problème est représenté en termes d'inconnues (variables de décisions) et de contraintes (expressions logiques) que doivent satisfaire les valuations des variables. Autrement dit, les inconnues d'un problème sont représentées par des variables contraintes (constrainted variables). Ainsi, pour un problème donné, la modélisation du problème consiste à déclarer les variables, définir les contraintes qui relient les variables, et appliquer ces contraintes.
Résoudre un tel problème consiste à trouver une valeur pour chaque variable, vérifiant simultanément toutes les contraintes. Cependant, lorsque nous commençons à travailler sur un problème, nous ne pouvons pas savoir à l'avance si une solution pourra satisfaire toutes les contraintes du problème : trouver une solution revient plutôt à rechercher une solution.
Il est aussi possible, pour un tel problème, de rechercher les valeurs qui optimisent une fonction donnée, souvent appelée fonction de coût (ou fonction objectif) du problème. En fait, il peut très bien y avoir plusieurs solutions (c'est-à-dire, plus d'un ensemble de valeurs pour les variables, qui satisfont les contraintes du problème) et, en relation avec le problème, il peut y avoir un certain critère considérant qu'un ensemble de valeurs est plus approprié que tout autre ensemble. Cette caractéristique est généralement mesurée en termes de fonction de coût, qui peut être optimisée (minimisée ou maximisée).

La programmation par contraintes est partagée en deux activitées relativement distinctes

- la représentation du problème.
- la recherche d'une ou de plusieurs solutions.

3.1 La représentation du problème

La représentation du problème consiste à déclarer les inconnues et les contraintes du problème. La représentation est spécifique au domaine du problème considéré, et requiert un langage de programmation très explicite pour traiter cette spécificité.
L'utilisation d'un langage orienté-objet, comme le C++, rend cette activité plus facile. `IlogSolver` est une bibliothèque de C++, qui implante la programmation par contraintes. En Solver, les objets de base sont appelés des variables contraintes. Pour chacune de ces variables, nous associons un ensemble de valeurs possibles, appelé domaine de la variable. Quand le domaine d'une variable ne contient plus qu'une seule valeur, on dit que la variable est fixée (ou liée). Les variables sont reliées entre elles par des contraintes. Beaucoup de contraintes

sont définies de manière générique (exemple : `m.add(IlcAllDiff(IlcAnyVarArray Tab))` est une contrainte qui impose que toutes les variables du tableau Tab sont deux à deux différentes). Il est aussi possible de créer ses propres contraintes ou plutôt ses propres classes de contraintes.
De manière générale, une contrainte est définie par deux fonctions : `void post()` et `void propagate()`. La fonction `post()` définit les évènements pour lesquels la contrainte est réveillée : ce sont généralement des évènements liés à des modifications du domaine de la variable : réduction du domaine (`void whenDomain`), changement de la borne minimum ou maximum du domaine(`void whenRange`), instantiation (`void whenValue`). Notons que si elle n'est déclarée qu'une seule fois, la contrainte est réveillée autant de fois qu'un évènement, défini dans la fonction `post`, se produit. La fonction `propagate()` définit comment les domaines des variables doivent être réduits par la contrainte, en éliminant du domaine des variables toutes les valeurs qui ne peuvent satisfaire la contrainte. Chaque type de contrainte possède son propre algorithme spécifique pour se propager. Des contraintes de types divers peuvent travailler ensemble : la communication entre les différents algorithmes de propagation s'accomplit à travers le domaine des variables. Cette propagation est réalisée sur l'ensemble des variables concernées par la contrainte, jusqu'à ce que le système arrive à un état stable, c'est-à-dire, un état où il n'y a plus de réduction possible de domaine.

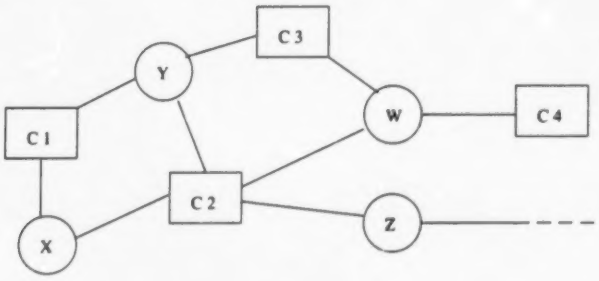

Figure 3.1: La propagation des contraintes

Sur l'exemple de la figure 3.1, les variables X, Y, Z et W sont soumises aux diverses contraintes $C1, C2, C3$ et $C4$. Si le domaine de la variable X est modifié, cela va provoquer le réveil des deux contraintes $C1$ et $C2$. Elles vont ensuite se propager sur les variables Y (qui est soumise aux deux contraintes), Z et W (ces deux variables étant soumises à la contrainte $C2$).

3.2 La recherche de solution

Résoudre un problème consiste à sélectionner une valeur dans le domaine de chaque variable, de sorte que toutes les contraintes soient satisfaites. Les variables du problème sont instantiées selon un algorithme de recherche préalablement défini de manière générique ou spécialisé. Elles peuvent être modélisées comme un arbre, où chaque noeud représente la fixation d'une variable. Par exemple, considérons deux variables qui prennent toutes les deux leur valeur dans l'ensemble {1;2;3}. L'ensemble des solutions sera représenté de la manière suivante (voir fig. 3.2).

La recherche d'une solution dans le graphe dépend de l'algorithme de recherche utilisé pour parcourir le graphe. Elle est basée sur deux principes :

- l'heuristique du choix de variable : elle définit une stratégie pour choisir une des variables parmi l'ensemble des variables n'étant pas encore instantiées. Cette heuristique peut dépendre de la taille du domaine des

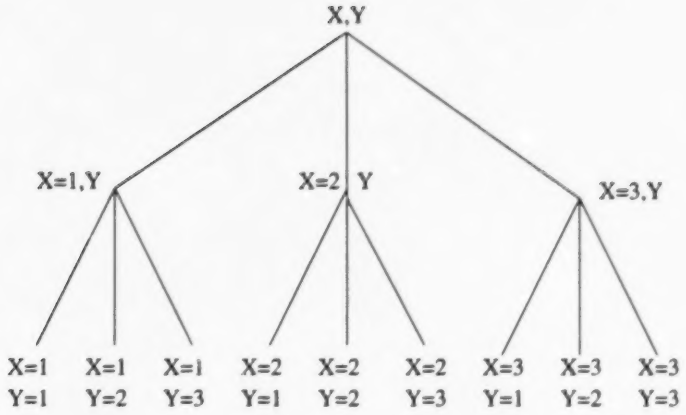

Figure 3.2: Représentation des solutions pour (X,Y)

variables (`IlcChooseMinSize` : stratégie qui choisit la variable ayant le plus petit domaine) ou bien d'un critère d'optimisation (choisir une variable qui va minimiser une certaine fonction).

- l'heuristique du choix de valeur : elle définit une stratégie pour choisir une des valeurs parmi l'ensemble des valeurs restantes pour instantier la variable sélectionnée. De même, plusieurs critères peuvent entrer en jeu pour déterminer le choix de la valeur.

La stratégie la plus simple et la plus générique est la lecture lexicographique : les variables sont instantiées dans un ordre bien précis, et, pour chaque variable, la valeur choisie est la plus petite du domaine. Ainsi, pour l'exemple ci-dessus du couple (X,Y), si nous instantions d'abord la variable X, puis la variable Y, nous aurons 9 solutions énoncées dans cet ordre : (X=1,Y=1), (X=1,Y=2), (X=1,Y=3), (X=2,Y=1), ...,(X=3,Y=3). Cette stratégie est efficace sur un problème de petite taille, mais pour des problèmes plus complexes, il est préférable de définir des stratégies de recherche plus élaborées.

3.2.1 L'action des contraintes

Pour rechercher des solutions, nous pourrions utiliser une première méthode qui consiste à "parcourir et vérifier": il s'agit d'énumérer toutes les combinaisons possibles des variables, et vérifier celles qui satisfont les contraintes (il est fort probable que toute combinaison n'est pas une solution). Cette méthode, efficace sur un problème de petite taille, est aussi coûteuse en calcul.
Mais la méthode préconisée par la programmation par contraintes est différente : après avoir appliqué les contraintes sur les variables, nous appliquons l'algorithme de recherche sur le graphe, et à chaque modification des domaines des variables, celui-ci restreint sa recherche en éliminant des arcs du graphe.
Par exemple, considérons l'ensemble des couples (X,Y), X et Y étant définis ci-dessus, et la contrainte $X \neq Y$. Quand la variable X va être instantiée, le domaine de Y sera systématiquement réduit en ôtant de son domaine, la valeur prise par la variable X. Grâce à l'action de cette contrainte, certains noeuds de l'arbre de recherche ne seront pas visités lors du parcours de l'arbre (voir figure 3.3).

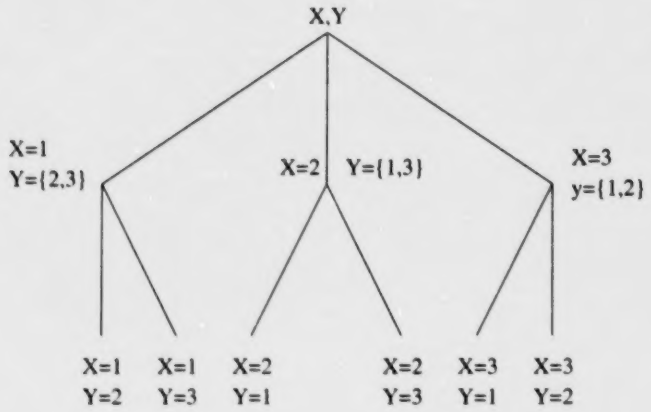

Figure 3.3: Chemin dans le graphe avec X≠Y

3.2.2 L'échec, et le retour arrière (backtracking)

Lors du parcours du graphe, il se peut qu'à un certain noeud, l'algorithme ne puisse plus instantier une variable (son domaine étant réduit à l'ensemble vide ou bien le choix d'une valeur ne satisfaisant plus les contraintes). Nous nous trouvons alors dans une situation d'échec : tout chemin défini à partir du chemin initialement pris ne peut conduire à une solution. L'algorithme revient donc en arrière (backtracking), choisit une autre valeur pour la variable considérée, tout en gardant le chemin initialement suivi, et continue le parcours du graphe jusqu'à une solution (toutes les variables sont instantiées et les valeurs satisfont les contraintes), ou une autre situation d'échec (voir fig. 3.4).

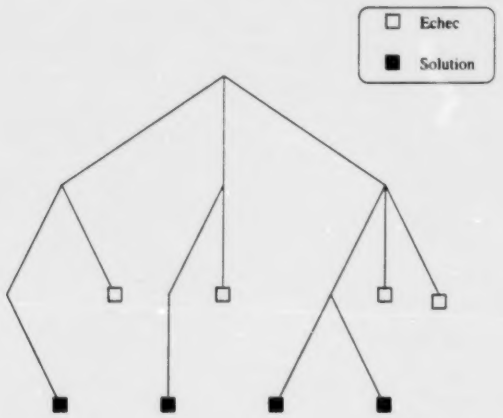

Figure 3.4: graphe des échecs et des solutions

Chapitre 4

Implantation du problème

Le modéle est implanté à l'aide du logiciel IlogSolver, qui utilise des librairies du langage C++. Ce langage est très efficace pour traiter des problèmes avec la programmation par contraintes.

4.1 La lecture et la modélisation des données

A partir du module de lecture des données développé par Pascal Labit et Tsévi Vovor, nous récupérons les données dans un objet de la classe `MData`. Soit `Pb`, une instance de la classe `MData`. Les membres privés de `Pb` sont les informations concernant la taille du problème (nombre d'infirmières, nombre de jours dans l'horizon,...,etc), ainsi qu'un vecteur d'objets de la classe `MNurse` (informations personnelles sur les infirmières), un vecteur d'objet de la classe MShift (informations sur les quarts de travail) et un vecteur d'objets de la classe MQuotaConstraints (information sur les contraintes de quota).

4.2 Choix de la modélisation

L'un des premières étapes dans la résolution du problème consiste à définir les variables de décisions. Cependant, les outils proposés par IlogSolver sont assez élaborés pour nous donner plusieurs options sur la modélisation et le choix des variables. Deux choix de modélisation s'offrent à nous : un modèle centré sur les infirmières (nurse-centered model) et un modéle centré sur les quarts (shift-centered model). Si nous avons retenu le second, il est intéressant d'évoquer les forces et les faiblesses du premier.

4.2.1 Le modèle centré sur les infirmières

Pour tout quart $s \in \mathcal{S}$, et pour tout jour $d \in \mathcal{D}$, nous associons un ensemble d'infirmières. Le modèle peut donc se modéliser ainsi :

$$\begin{array}{ccc} \mathcal{S} \times \mathcal{D} & \longrightarrow & \mathcal{N} \\ (s,d) & \mapsto & X_{s,d} \end{array}$$

Ici, X est une variable ensembliste (`IlcAnySetVar`) : le domaine associé à cette variable est un ensemble d'ensembles. IlogSolver représente ce domaine par ses limites supérieures et inférieures. La limite supérieure est l'union de toutes les valeurs possibles de la variable, c'est-à-dire, l'union de tous les ensembles-éléments du domaine. La limite inférieure est l'intersection de toutes les valeurs possibles du domaine, c'est-à-dire

l'intersection de tous les ensembles-éléments du domaine.
Le domaine de la variable ensembliste est représenté par deux ensembles :

- l'ensemble des éléments requis (required set), c'est-à-dire l'ensemble des éléments qui appartiennent à toutes les valeurs possibles de la variable (la limite inférieure).
- l'ensemble des éléments possibles (possible set), c'est-à-dire l'ensemble des éléments qui appartiennent à au moins une des valeurs possibles de la variable (la limite supérieure).

Lors de l'initialisation de la variable, l'ensemble des éléments requis est inclus dans l'ensemble des éléments possibles. Lorsqu'une variable est fixée, les élements requis sont les mêmes que les éléments possibles, et ils constituent la valeur de la variable.

$$\begin{array}{lcccl} \text{Initialisation} & \text{required set} & \supseteq & X & \supseteq \text{ possible set} \\ \text{Variable fixée} & \text{required set} & = & X & = \text{ possible set} \end{array}$$

L'expression des contraintes Si certaines contraintes pouvaient se modéliser de manière très simple, comme les pré-affectations, d'autres semblaient plus difficile à cerner, car elles faisaient souvent appel à de nombreuses intersections d'ensembles. Par exemple, considérons la contrainte dure sur les charges de travail. Avec les arguments vus dans la modélisation du problème, la contraintes s'exprime de la manière suivante avec des variables ensemblistes : Pour tout infirmière $n \in \mathcal{N}$, pour toute fin de semaine $w \in \mathcal{W}_n$,

$$minHoursPaid_{n,w} \leq \sum_{s \in \mathcal{S}} \sum_{d=startingDay_{n,w}}^{endingDay_{n,w}} HoursPaid(X_{s,d} \cap \{n\}) \leq maxHoursPaid_{n,w}$$

Globalement, la plupart des contraintes s'exprimaient sous la forme d'unions et d'intersections d'ensembles. Quand nous commençions à réflechir sur les moyens d'implantation de ce modèle avec `IlogSolver`, nous étions rapidement confronté à certaines impasses (complexité des calculs, manque des fonctions sur les variables ensemblistes,...,etc). Après un mois de réflexion sur le modèle centré sur les infirmières, celui-ci a été abandonné au profit du modèle centré sur les quarts.

4.2.2 Le modèle centré sur les quarts

Pour tout couple $(n, d) \in \mathcal{N} \times \mathcal{D}$, nous définissons une variable de décision qui prend ses valeurs dans l'ensemble $\mathcal{S}$. Toutes nos variables de décision sont des variables à domaine fini, puisque l'ensemble $\mathcal{S}$ est de domaine fini. Pour représenter nos variables de décision, nous utilisons la structure de données `IlcAnyVarArray* Tab_Var`, qui est un tableau de variables quelconques (entiers, chaines de caractères,...,etc)

```
// Declaration du tableau des variables de decisions
// C'est un tableau a deux entrees : N*D
IlcIntVarArray *Tab_Var;
Tab_Var = new (m.getHeap()) IlcIntVarArray[N];
for(IlcInt n = 0 ; n < N ; n++ )
  Tab_Var[n] = IlcIntVarArray(m, D, 0, S+1);
```

Remarque : nous utilisons en fait la structure de données `IlcIntVararray` qui est un tableau de variables à valeurs entières.
Chaque variable de ce tableau prend ses valeurs dans l'ensemble $\mathcal{S} = \{0, 1, \ldots, S-1\}$ des quarts réalisables

($S = card(\mathcal{S})$). A cela, nous ajoutons les valeurs S et $S+1$, représentant respectivement les quarts non travaillés ("Off") et les quarts de congés ("Vacation"). Ainsi, pour tout $n \in \mathcal{N}$ et $d \in \mathcal{D}$, si `Tab_Var[n][d]` n'est pas réduit à un singleton, il représente l'ensemble des quarts susceptibles d'être travaillés par l'infirmière n le jour d de l'horizon. Sinon, `Tab_Var[n][d]` est le quart travaillé par l'infirmière n le jour d de l'horizon.

4.3 L'implantation des contraintes

4.3.1 Les contraintes dures

Les pré-affectations et les affectations interdites

```
void mC_PreAssignments(IlcManager m, MData* Pb, IlcIntVarArray* TabVar);
void mC_ForbiddenAssignments(IlcManager m, MData* Pb, IlcIntVarArray* TabVar);
```

Nous utilisons des contraintes d'égalités (pré-affectations) et de non-égalité (affectations interdites) pour exprimer ces contraintes. Pour les affectations, si la valeur indiquée n'appartient pas au domaine de la variable évoquée, un échec est renvoé à l'exécution.
A tout objet MNurse, sont associées les vecteurs `mListPreAssignments` et `mListForbiddenAssignments`, contenant respectivement des objets `MPreAssignments` et `MForbiddenAssignments`. Chaque objet a pour membres privés un jour `mDay` et un quart `mShift`. Les contraintes sont implantées de la manière suivante :

Pré-affectations

```
// Boucle sur les infirmieres : pour tout n=[0...N-1]
for(IlcInt n = 0 ; n < Pb->mGetNbNurses() ; n++)
//Boucle sur le la taille du vecteur des PreAssignments : pour tout c=[0...C-1]
for(IlcInt c = 0 ; c < Pb->mGetListNurses()[n]->mGetListPreAssignments().size() ; c++)
// Contrainte Tab_Var[n][d]==[s]
TabVar[n][Pb->mGetListNurses()[n]->mGetListPreAssignments()[c]->mGetDay()]
== (Pb->mGetListNurses()[n]->mGetListPreAssignments()[c]->mGetShift());
```

Affectations interdites

```
// Boucle sur les infirmieres : pour tout n=[0...N-1]
for(IlcInt n = 0 ; n < Pb->mGetNbNurses() ; n++)
//Boucle sur le la taille du vecteur des ForbiddenAssignments : pour tout c=[0...C-1]
for(IlcInt c = 0 ; c < Pb->mGetListNurses()[n]->mGetListForbiddenAssignments().size() ; c++)
// Contrainte Tab_Var[n][d] != [s]
TabVar[n][Pb->mGetListNurses()[n]->mGetListForbiddenAssignments()[c]->mGetDay()]
!= (Pb->mGetListNurses()[n]->mGetListForbiddenAssignments()[c]->mGetShift());
```

Il existe une autre manière d'opérer, mais Elle agit plus comme une action d'instantiation qu'une action de contrainte. Certes, leur application va provoquer des réductions de domaine qui vont réveiller les contraintes liées aux variables réduites. Mais la propagation des autres contraintes n'auront aucun effet sur les pré-affectations et les affectations interdites.
Pour les pré-affectations, nous utilisons la fonction `void setValue(IlcAny Value) const`, qui affecte Value à la variable considérée. Si Value n'était pas dans le domaine de la variable, un échec est renvoyé à l'exécution.

Pour les affectations interdites, nous utilisons la fonction **void setValue(IlcAny Value) const**. Elle élimine la valeur Value de l'ensemble des valeurs possibles de la variable considérée. Si le retrait de value rend vide le domaine des valeurs possibles de la variable, un échec est renvoyé à l'exécution.
Nous écrirons alors :

```
// Pre-affectations
TabVar[n][Pb->mGetListNurses()[n]->mGetListPreAssignments()[c]->mGetDay()]
.setValue(Pb->mGetListNurses()[n]->mGetListPreAssignments()[c]->mGetShift());
\\ Affectations interdites
TabVar[n][Pb->mGetListNurses()[n]->mGetListForbiddenAssignments()[c]->mGetDay()]
.removeValue(Pb->mGetListNurses()[n]->mGetListForbiddenAssignments()[c]->mGetShift());
```

Les quarts réalisables

```
void mC_FeasibleShifts(IlcManager m, MData* Pb, IlcIntVarArray* Tab);
```

Pour chaque instance de la classe MNurse, est défini un ensemble de quarts réalisables par l'infirmière. Nous utilisons la fonction **void setDomain(IlcAnySet set)**. Elle restreint le domaine de la variable faisant appel à cette fonction, à l'intersection entre celui-ci et l'ensemble **set** appelé en argument : toutes les valeurs du domaine n'appartenant pas à l'ensemble **set** seront éliminées du domaine. Si le domaine de la variable devient vide, un échec est renvoyé à l'exécution.

```
  // Boucle sur les infirmieres : pour tout n dans [0...N-1].
  for(IlcInt n = 0 ; n < Pb->mGetNbNurses() ; n++)
  {
    // SetFeasibleshift : Declaration d'un ensemble d'entiers
    // de taille S+2, non initialise (IlcFalse).
    IlcIntSet SetFeasibleShift(m, 0 , Pb->mGetNbFeasibleShifts() + 1, IlcFalse);

    // Construction de cet ensemble (application de void add(IlcInt Value)).

    // Boucle sur les jours de l'horizon
    for(IlcInt d = 0 ; d < Pb->mGetNbDaysInHorizon() ; d++)
      Tab[n][d].setDomain(SetFeasibleShift);
  }
```

Les quarts du lendemain

```
void mC_NextDayShifts(IlcManager m, MData* Pb, IlcIntVarArray* Tab, IlcIntArray TabShift);
```

Pour chaque quart de travail, nous construisons un ensemble d'entiers qui identifient les quarts réalisables au lendemain du quart évoqué.

```
IlcIntSet*  mSetNextDayShifts = new (m.getHeap()) IlcIntSet [S+2];
for(IlcInt s = 0 ; s < S ; s++)
  mSetNextDayShifts[s] = IlcIntSet(m, TabShift, IlcFalse);
  \\ Construction de l'ensemble a partir des donnees.
  ... mSetNextDayShifts[s].add(Value);
```

Pour tout $n \in \mathcal{N}$ et $d \in \mathcal{D}\backslash\{\text{dernier jour}\}$, nous voulons appliquer une contrainte qui prends en compte un couple de variables de décision (`Tab_Var[n][d]`,`Tab_Var[n][d+1]`). Pour cela, nous allons appliquer une contrainte de la manière suivante :

```
for(IlcInt n = 0 ; n < Pb->mGetNbNurses() ; n++)
  for(IlcInt d = 0 ; d < Pb->mGetNbDaysInHorizon() ; d++)
    m.add(mConstraintNextDayShifts(Tab, d, n, mSetNextDayShifts));
```

`mConstraintNextDayShifts` est une contrainte qui retourne une instance de la classe `ClassNextDayShifts`, dérivant de la class IlcConstraintI.

```
IlcConstraint mConstraintNextDayShifts(IlcIntVarArray* Tab,
       IlcInt d, IlcInt n,
       IlcIntSet* setNextDayShifts)
{
IlcManager m = Tab[n][d].getManager();
return new (m.getHeap()) ClassNextDayShifts(m, Tab, d, n, setNextDayShifts);
}
```

Il nous reste maintenant à définir les méthodes `post` et `propagate` de la contrainte. La méthode `post` définit les conditions de réveil : la contrainte est réveillée lorsque le domaine de la première variable du couple est liée.

```
void ClassNextDayShifts::post()
{
  Tab[n][d].whenValue(this);
}
```

La méthode `propagate` définit les évènements liés à la propagation de la contrainte : elle fait l'intersection entre le domaine de la seconde variable du couple, et l'ensemble des quarts réalisables au lendemain.

```
void ClassNextDayShifts::propagate()
{
IlcInt val;
val= (Tab[n][d].getValue());
Tab[n][d+1].setDomain(setNextDayShifts[val]);
}
```

Les charges de travail

```
void mC_WorkLoad(IlcManager m, MData* Pb, IlcIntVarArray* Tab, IlcIntArray TabHours);
```

Pour chaque infirmière n et pour chaque contrainte de charge de travail wkl de l'infirmière n, nous récupérons dans un tableau de variables toutes les variables correspondant aux quarts travaillés (ou non) par l'infirmière n, pendant la période définie par les dates de début et de fin de la contrainte de charge de travail wkl.
Pour calculer les charges de travail, nous convertissons les variables correspondant aux quarts, en variables correspondant aux temps de travail des quarts. Soit `TabHours`, un tableau d'entiers, où `TabHours[i]` est le temps de travail payé du quart i (nous convertirons tous les temps en minutes, les calculs n'en seront que plus maléables). Si nous considérons la variable `Tab_Var[n][d]`, dont le domaine n'est pas réduit à une unique valeur, alors `TabHours[Tab_Var[n][d]]` est une variable dont le domaine contient les temps de travail des

quarts présent dans le domaine de la variable Tab_Var[n][d], comme le montre l'exemple qui suit.
Exemple de **TabHours** :

Shift	Time (min)
Day	480(8*h*)
Evening	450(7*h*30)
Night	720(12*h*)
Off	0

Si X = {Day,Night,Off}, alors $TabHours[X] = \{480, 720, 0\}$

Nous avons donc un tableau de variables **TabWorkLoad**, dont les variables sont étroitement liées aux variables de Tab_Var.

```
IlcIntVarArray **TabWorkLoad;
TabWorkLoad = new (m.getHeap()) IlcIntVarArray*[N];
for(IlcInt n = 0 ; n < Pb->mGetNbNurses() ; n++)
{
  TabWorkLoad[n] = new (m.getHeap()) IlcIntVarArray[WKL];
  IlcInt Periode= endDay-startDay+1;
  TabWorkLoad[n][wkl] = IlcIntVarArray(m,Periode);
  for(IlcInt t = 0 ; t < Periode ; t++)
    TabWorkLoad[n][wkl][t] =tabHours[Tab[n][t+startDay]];
}
```

Pour obtenir la charge de travail de la période sur laquelle porte la contrainte, nous utilisons la fonction **IlcSum**, qui renvoie une variable égale à la somme des variables du tableau, que nous bornons par les charges minimales et maximales indiquées par les données de la contrainte.

```
m.add(IlcSum(TabWorkLoad[n][wkl]) <= maxHours);
m.add(IlcSum(TabWorkLoad[n][wkl]) >= minHours);
```

Les quotas

```
void mC_Quota(IlcManager m, MData* Pb, IlcIntVarArray* TabVar);
```

Cette contrainte est une boucle sur l'ensemble des instances de la classe **MQuotaConstraints**, se trouvant dans le vecteur **mListQuota** de l'instance **Pb**. Ici **Ctr** représente l'instance de la classe **MQuotaConstraints** en cours de traitement.

```
//Boucle sur la taille de mGetListQuotaConstraints().Size()
for(IlcInt index = 0 ; index < Pb->mGetListQuotaConstraints().size() ; index++)
  {
    MQuotaConstraints* Ctr = Pb->mGetListQuotaConstraints()[index];
    ...
  }
```

La première étape consiste à récupérer les niveaux requis pour la contrainte de quota.

```
IlcIntSet SetSkillLevel(m, 1, L, IlcFalse);
for(IlcInt i=0;i< Ctr->mGetListSkillLevel().size();i++)
  SetSkillLevel.add(Ctr->mGetListSkillLevel()[i]);
```

Nous identifions les variables correspondant aux infirmières ayant le niveau requis, et travaillant le jour évoqué par la contrainte de quota.

```
// WorkingNurses : Declaration d'un tableau de variables qui va recuperer
// tous les quarts travailles par les infirmieres de niveau l
// (appartenant a la liste des niveaux de la contrainte), le jour d de la contrainte.
IlcIntVarArray WorkingNurses(m,N,0,S+1);
IlcInt index=0;
for(IlcInt n=0;n<N;n++)
  if(SetSkillLevel.isIn(Pb->mGetListNurses()[n]->mGetSkillLevel()))
    WorkingNurses[index++]=TabVar[n][Ctr->mGetDay()];
```

Nous déclarons un tableau d'entiers qui va récupérer tous les identifiants entiers des quart évoqués par la contrainte de quota.

```
// Values : Tableau qui repertorie tous les quarts qui
// couvrent la periode de la contrainte.
IlcIntArray Values(m,Ctr->mGetListShifts().size());
for(IlcInt i=0;i<Ctr->mGetListShifts().size();i++)
  Values[i]=Ctr->mGetListShifts()[i].mGetIdentifiant();
```

Nous déclarons un tableau de variables entières qui va contenir les cardinaux des valeurs de **Values**, contenues dans le tableau **WorkingNurses**

```
// Card : Declaration d'un tableau de variables. Card[i] est
// le nombre d'occurences de Values[i] dans WorkingNurses.
// La taille de Card est la meme que celle de Values.
// Chaque valeur de TabCard est comprise entre 0 et compteur (nb max d'inf).
IlcIntVarArray Card(m,Values.getSize(),0,N);
```

Nous distribuons les valeurs de **WorkingNurses** dans **Card**. Pour se faire, nous utilisons la fonction **IlcDistribute** dont la syntaxe est la suivante :

```
IlcConstraint IlcDistribute (IlcIntVarArray Card,
     IlcAnyArray Values,
     IlcAnyVarArray Vars)
```

Quand la contrainte est activée, les variables du tableau **Card** sont égales aux nombres d'occurences dans le tableau **Vars**, des valeurs **values** du tableau **Values**. Plus précisemment, pout tout i, **Card[i]** est égale au nombre d'occurence de **Values[i]** dans le tableau **Vars**.

```
m.add(IlcDistribute(Card,Values,WorkingNurses));
```

Nous appliquons la contrainte : min $\leq$ effectif $\leq$ max.

```
m.add(effectif <= Ctr->mGetQuotaTarget() + Ctr->mGetQuotaSurplus());
m.add(effectif >= Ctr->mGetQuotaTarget() - Ctr->mGetQuotaDeficit());
```

Les congés

```
void mC_Vacations(IlcManager m, MData* Pb, IlcIntVarArray* TabVar);
```

Pour chaque infirmière, nous vérifions la valeur du "drapeau" pour connaitre l'état d'attribution de jours de congé pour l'infirmière n présentemment traitée. Si cette état est inactif (valeur du drapeau nulle), alors la valeur entière correspondant au quart "vacations" est ôtée des domaines des variables de l'infirmière n

```
for(IlcInt n=0;n<N;n++)
{
MNurse* Ctr=Pb->mGetListNurses()[n];

if(Ctr->mGetActiveRuleHolidaysAndVacationsToGrand()==0)
  for(IlcInt d = 0 ; d < D ; d++)
    m.add(TabVar[n][d] != S+1);
...
```

Sinon, nous récupérons les variables de décision correspondants aux jours compris entre les jours de début et de fin définis dans les données de l'infirmière n. Sur cette cette période, nous appliquons la fonction de distribution, afin de récupérer le nombre (variable) de quarts "vacations" durant cette période. Nous appliquons ensuite la contrainte en bornant cette variable par les nombres minimal et maximal de jours de congé à accorder pour l'infirmière n.

```
IlcInt period = Ctr->mGetVacationsLatestDay()-Ctr->mGetVacationsEarliestDay()+1;
IlcIntVarArray TabVacations(m,period,0,S+1);
for(IlcInt i=0;i<period;i++)
  TabVacations[i]=TabVar[n][i+Ctr->mGetVacationsEarliestDay()];
IlcIntVarArray Card(m,1,0,period);
IlcIntArray Value(m,1,S+1);
m.add(IlcDistribute(Card,Value,TabVacations));
m.add(Card[S+1] >= Ctr->mGetVacationsMinNbDaysOff());
m.add(Card[S+1] <= Ctr->mGetVacationsMaxNbDaysOff());
```

Les quarts consécutifs

```
void mC_ConsecutiveShifts(IlcManager m, MData* Pb, IlcIntVarArray* Tab);
```

La contrainte sur les quarts consécutifs est une contrainte globale sophistiquée, dont la description dépasse le cadre du rapport. Elle a été développée par Gilles Pesant pour un projet sur les horaires cycliques. L'idée générale est la suivante : quand le domaine d'une variable est modifié, nous allons regardé le voisinage de cette variable. Si nous pouvons déceller une suite maximale de quarts consécutifs d'un même type t (voir figure 4.1), alors nous pouvons propager la contrainte en "brisant"la suite, c'est-à-dire, en ôtant les quarts de type t des variables situés aux frontières de la suite (voir figure 4.2). De même, il est possible de briser la suite de l'intérieur si nous savons que l'instantiation d'une valeur d'une certain type produira une trop grande suite de quarts consécutifs du même type (voir 4.3). L'implantation de cette contrainte n'est que l'appel d'une fonction appelée `ValidSeqNonRot` par Gilles Pesant.

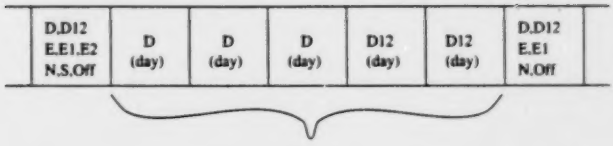

Figure 4.1: quarts consécutifs de type *day*, max=5

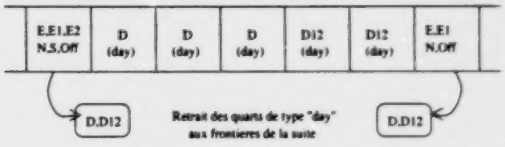

Figure 4.2: Propagation de la contrainte des quarts consécutifs

Les fins de semaines consécutives

```
void mC_ConsecutiveWeekEnds(IlcManager m, MData* Pb, IlcIntVarArray* Tab);
```

Nous utilisons la contrainte des quarts consécutifs, mais sur un tableau de variables auxiliaires, liées aux différents variables de décision correspondant aux fins de semaines de l'horizon. Il y aura donc autant de variables auxiliaires qu'il y a de fins de semaine dans l'horizon (attention : un dimanche seul au début de l'horizon, ou bien un samedi seul en fin d'horizon, compte comme une fin de semaine). Les variables auxiliaires sont contenues dans un tableau **TabWeekEnd**, et prennent la valeur 0 (si la fin de semaine n'est pas travaillée), et 1 sinon.
Les variables de **Tab_Var** et **TabWeekEnd** sont liées par des contraintes (voir figure 4.4), que nous implantons à l'aide des méthodes **post** et **propagate**.

- **post** : Les domaines des variables de décision ou auxiliaires sont modifiés.
- **propagate** : Nous modifions les domaines des variables en fonction des évènements induits. Par exemple, si la variable auxiliaire est liée à la valeur 0 (fin de semaine non travaillée), alors nous pouvons enlever tous les quarts de travail dans les variables de décision qui sont associées à cette variable auxiliaire.

Enfin, nous appliquons la fonction ValidSeqNonRot (fonction qui applique la contrainte des quarts successifs) au tableau des variables auxiliaires.

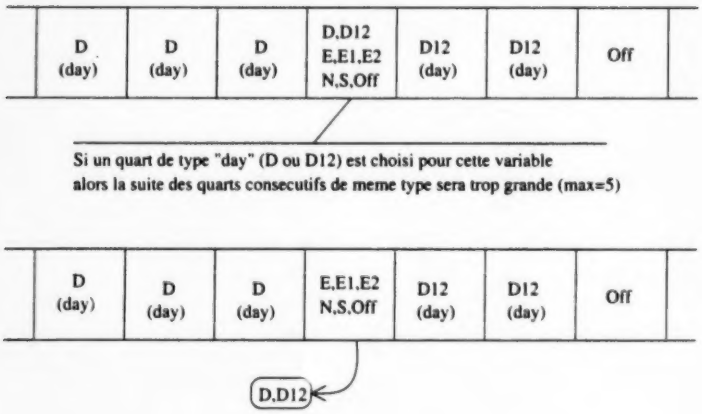

Figure 4.3: suite brisée de l'intérieur

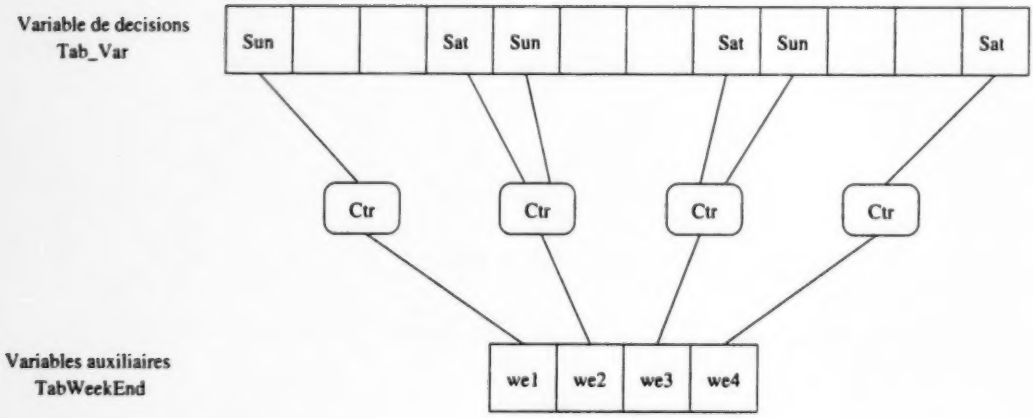

Figure 4.4: Liaison entre les variables auxiliaires et les variables de décision

Chapitre 5

Stratégies et résultats

5.1 Heuristique de choix de variable

5.1.1 L'ordre lexicographique

Pour générer une solution, les variables doivent être rangées dans un vecteur (`IlcIntVarArray`). Lors de la recherche d'une solution (`m.add(IlcGenerate(IlcIntVarArray TabVar)`, le Solveur d'`IlogSolver` instantie les variables dans l'ordre pré-défini dans le vecteur. En effet, il est déjà possible d'orienter notre stratégie de recherche sur l'ordre d'instantiation : par exemple, nous pouvons choisir d'instantier les variables infirmière par infirmière, ou bien jour par jour (voir figure 5.1). L'ordre peut revêtir une certaine importance dans le

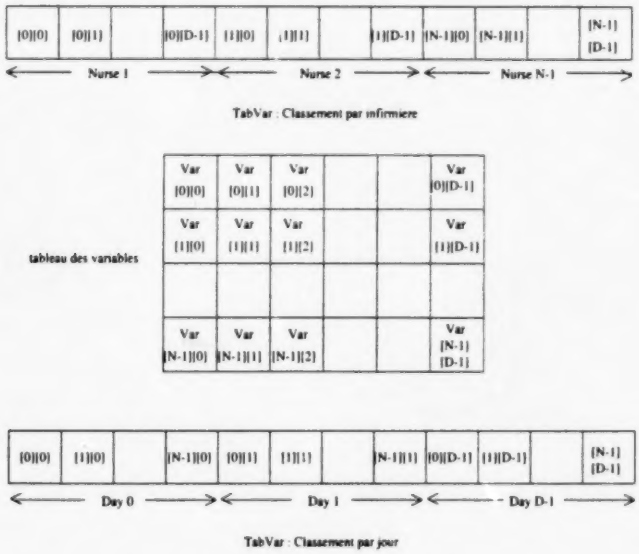

Figure 5.1: stratégie d'instantiation des variables

traitement des contraintes. En effet, si nous instantions nos variables par infirmière, la contrainte des charges de travail sera réveillée de nombreuses fois avant d'autres contraintes, puisque nous instantions les variables correspondant à l'horizon complet de la première infirmière, puis de la seconde,...,etc. Par contre, si nous instantions nos variables par jour, nous réveillerons plus souvent les contraintes sur les quotas.

5.1.2 Choix du plut petit domaine

Quand une variable est instantiée, les contraintes se propagent sur les autres variables en réduisant leur domaine si besoin. Si leur domaine est plus court, le choix d'instantiation est plus réduit, et la probabilité de trouver une solution est plus grande (il y a moins de noeuds à parcourir). A lieu de choisir la première variable qui suit la variable instantiée, nous pouvons choisir la première variable dont le domaine est le plus réduit, et qui suit la variable instantiée. Cette stratégie est implantée par la commande `m.add(IlcGenerate(IlcIntVarArray, IlcChooseMinIndex))`.

5.2 Heuristique de choix de valeur

5.2.1 L'ordre lexicographique

Par défaut, le solveur choisit la plus petite valeur parmi celles restant dans le domaine de la variable. Par la suite, nous verrons que cela crée un phénomène de redondance dans la solution (voir résultats).

5.2.2 Le choix aléatoire

Nous avons vu précédemment que la lecture lexicographique engendrait une certaine redondance au niveau de l'instantiation des quarts pour les premières variables, et que cela créait de nombreux "backtracking". Pour enrayer ce phénomène, nous avons légèrement modifié la stratégie. Si l'heuristique de choix de variable est toujours basé sur l'ordre lexicographique du tableau (par infirmière ou par jour), l'heuristique de choix de valeur devient aléatoire : une valeur parmi le domaine est choisie au hasard pour être instantiée à la variable.

5.3 Test expérimentaux

Nous avons tester le programme sur deux fichiers de données : le premier fichier est un jeu de données expérimentales, il provient du CHUM, et comporte 7 infirmières avec un horizon de 4 semaines. Le second fichier provient de l'hôpital Royal Victoria, et propose les conditions réelles d'un service de maternité avec 50 infirmières, avec un horizon 6 semaines.

5.3.1 L'ordre lexicographique

Pour tester le programme, nous disposons au préalable d'un fichier de données sur 7 infirmières. Voici un exemple de la première solution générée par l'ordre lexicographique sur 7 jours, l'instantiation se faisant infirmière par infirmière).

Recherche de solution :

nurse	name	J1	J2	J3	J4	J5	J6	J7
1	xx	JOUR	JOUR	JOUR	JOUR	JOUR	Off	Off
2	xx	JOUR	vac	JOUR	JOUR	JOUR	JOUR	Off
3	xx	JOUR	JOUR	JOUR	JOUR	JOUR	Off	Off
4	xx	NUIT	NUIT	NUIT	NUIT	Off	Off	JOUR
5	xx	SOIR	SOIR	SOIR	Off	Off	JOUR	JOUR
6	xx	Off	Off	NUIT	NUIT	NUIT	NUIT	NUIT
7	xx	Off	Off	JOUR	SOIR	SOIR	SOIR	SOIR

Number of fails : 44774
Number of choice points : 44802
Number of variables : 1646
Number of constraints : 1061
Running time since creation : 25.63

Voici maintenant un autre exemple de première solution où l'instantiation se fait jour par jour :

Recherche de solution :

nurse	name	J1	J2	J3	J4	J5	J6	J7
1	xx	JOUR	JOUR	JOUR	JOUR	JOUR	Off	Off
2	xx	JOUR	vac	JOUR	JOUR	JOUR	JOUR	Off
3	xx	JOUR	JOUR	JOUR	JOUR	JOUR	Off	Off
4	xx	NUIT	NUIT	JOUR	Off	Off	JOUR	JOUR
5	xx	SOIR	SOIR	SOIR	SOIR	Off	Off	JOUR
6	xx	Off	Off	NUIT	JOUR	SOIR	SOIR	SOIR
7	xx	Off	Off	NUIT	NUIT	NUIT	NUIT	NUIT

Number of fails : 73235
Number of choice points : 73264
Number of variables : 1646
Number of constraints : 1061
Running time since creation : 46.56

Observations et analyses des résultats

Sur l'ensemble des deux solutions, nous remarquons qu'elles semblent similaires pour les trois premières infirmières, et que plusieurs changement s'opèrent au niveau des quatre dernières infirmières. De plus, nous constatons que le nombre d'échecs (`Number of fails`) et le temps d'exécution (`Running time since creation`) sont globalement élevés très nettement supérieurs pour la deuxième stratégie (jour). Enfin, nous pouvons remarquer qu'il y a une certaine répétition des quarts pour les premières infirmières (longues suites d'un même quart), alors que les compositions des horaires des dernières infirmières semblent plus ecclectiques.

A la lumière de ces observations, nous pouvons déjà donner une première conclusion : les deux stratégies ne sont pas excellentes, compte tenu des temps d'exécution et des nombres d'échecs.
Nous pouvons aussi donner des indications sur le comportement du solveur en fonction des contraintes. Le solveur instantie les premières variables avec les valeurs les plus basses. Comme le problème est peu contraint (toutes les infirmières peuvent travailler tous les quarts, contraintes faibles sur les quarts du lendemain,...,etc), la valeur la plus basse est très souvent la même pour plusieurs variables (pour la première infirmière, nous avons une série de 5 quarts "jour"). Dans les premières instantiations, les contraintes sur les quotas et les charges de travail ne sont que très rarement réveillées.
Si nous utilisons une stratégie de recherche infirmière par infirmière, la contrainte des charges de travail va intervenir lors de l'instantiation de la sixième variable (en moyenne, chaque infirmière doit travailler 4 à 5 quarts par semaine). Comme les premiers jours son occupées par des quarts de travail, des quarts de repos sont requis pour les derniers jours. Solveur réitère cette démarche pour les premières infirmières. Arrivé à une certaine variable, la contrainte de quota est réveillée. A ce moment, il y a une grande probabilité pour que la contrainte sur le nombre maximum d'infirmières, présentes pendant le quart de jour, soit saturée. Il va donc affecter les autres quarts restant de telle sorte que les minima des contraintes de quotas doivent être respectés. Comme le solveur a distribué uniformément des quarts "jour" dans la première partie de l'horizon, les choix des quarts restants sont réduits compte tenu de toutes les contraintes à satisfaire.
Lorsque le solveur est dans l'incapacité d'instantier une variable (domaine nul ou toutes les contraintes saturées), il renvoie un échec et commence le "backtracking" et prendre une autre direction dans l'arbre de recherche. Il est possible qu'il puisse trouver une solution dans une partie de l'arbre proche de la première direction. Dans ce cas-là, il y a des chances que le nombre d'échecs soit faible. En revanche, si les premières variables sont instantiées de manière qu'il ne puisse y avoir aucune solution réalisable, alors il y aura de très nombreux "backtrackings", et le solveur mettra beaucoup de temps à explorer toutes les parties de l'arbre, avant de remonter et prendre une autre direction (voir figure 5.2).
Les mêmes conclusions peuvent s'appliquer si nous choisissons la stratégie d'instantiation des variables jour après jour. Dans les premières instantiations, les contraintes de quota seront surtout mises à contribution. Le "backtracking interviendra surtout lorsque les contraintes de charge de travail seront réveillées. Cependant, les nombres supérieurs d'échecs et de temps d'exécution nous laissent à penser que les contraintes ont été réveillées beaucoup plus de fois que pour la première stratégie.

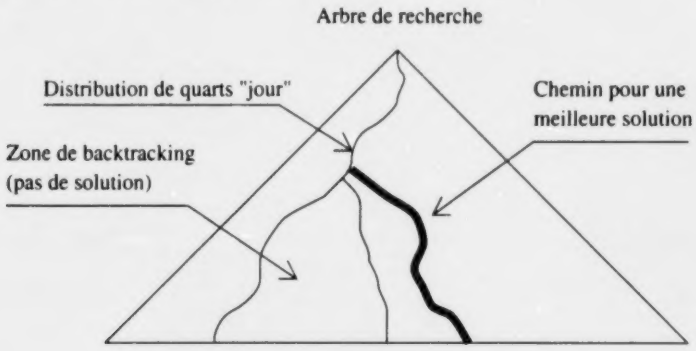

Figure 5.2: exemple de la zone de backtracking

5.3.2 Stratégie de choix aléatoire

Recherche de solution (lecture par infirmière)

nurse	name	J1	J2	J3	J4	J5	J6	J7
1	xx	Off	Off	JOUR	JOUR	SOIR	SOIR	SOIR
2	xx	NUIT	vac	SOIR	SOIR	SOIR	SOIR	Off
3	xx	SOIR	SOIR	Off	Off	JOUR	JOUR	JOUR
4	xx	Off	Off	Off	NUIT	NUIT	NUIT	JOUR
5	xx	JOUR	JOUR	SOIR	Off	Off	Off	NUIT
6	xx	NUIT	NUIT	NUIT	JOUR	JOUR	Off	Off
7	xx	JOUR	JOUR	JOUR	Off	Off	JOUR	Off

Number of fails	:	2130
Number of choice points	:	2155
Number of variables	:	1646
Number of constraints	:	1061
Running time since creation	:	4.5

Recherche de solution (lecture par jour)

nurse	name	J1	J2	J3	J4	J5	J6	J7
1	xx	Off	Off	JOUR	JOUR	SOIR	SOIR	SOIR
2	xx	NUIT	vac	SOIR	SOIR	SOIR	SOIR	Off
3	xx	SOIR	SOIR	Off	Off	JOUR	JOUR	JOUR
4	xx	Off	Off	Off	JOUR	JOUR	JOUR	JOUR
5	xx	JOUR	JOUR	JOUR	SOIR	Off	Off	NUIT
6	xx	JOUR	JOUR	JOUR	Off	Off	NUIT	Off
7	xx	NUIT	NUIT	NUIT	NUIT	NUIT	Off	Off

Number of fails	:	49
Number of choice points	:	72
Number of variables	:	1646
Number of constraints	:	1061
Running time since creation	:	0.5

Observations et analyses des résultats

Nous constatons que le nombre d'échecs et le temps d'exécution sont très inférieurs aux résultats précédents. Cela démontre que le phénomène de backtracking a été réduit. Comme pour la lecture lexicographique, nous constatons que les horaires proposés pour les premières infirmières sont similaires et que des changements s'opèrent pour les dernières infirmières. Par contre, nous pouvons observer que les résultats sur le nombre d'échecs et le temps d'exécution pour l'instantiation par jour est nettement inférieur à ceux pour l'instantiation par infirmière.

Pour les deux solutions proposées, les premiers quarts affectés aux premières infirmières ne sont pas des quarts "jour" : le phénomène aléatoire propose une plus grande diversité dans l'instantiation des premières variables. Cela empêche la saturation des contraintes de quotas lorsque le solveur commence à instantier les variables des dernières infirmières. De plus, il semble que ce phénomène de diversité facilite l'action des contraintes lors de l'instantiation des variables, puisque les contraintes ne commençent à être saturées que vers les niveaux inférieurs de l'arbre de recherche.

5.3.3 Test sur le fichier des 50 infirmières

Nous présentons maintenant un horaire généré pour 50 infirmières sur 7 jours.

Nurse	Name	*D*1	*D*2	*D*3	*D*4	*D*5	*D*6	*D*7
1	*xx*	*E*	*E*	*E*	*Off*	*Off*	*Off*	*Off*
2	*xx*	*N*	*N*	*N*	*N*	*N*	*N*	*Off*
3	*xx*	*N*	*N*	*N*	*Off*	*Off*	*N*	*N*
4	*xx*	*S*12	*S*12	*S*12	*Off*	*Off*	*S*12	*S*12
5	*xx*	*E*1	*E*1	*D*	*D*	*Off*	*E*	*E*1
6	*xx*	*D*	*D*	*vac*	*vac*	*vac*	*vac*	*vac*
7	*xx*	*S*12	*S*12	*vac*	*vac*	*vac*	*S*12	*S*12
8	*xx*	*D*	*D*	*D*	*E*	*Off*	*Off*	*D*
9	*xx*	*E*	*E*	*Off*	*Off*	*Off*	*Off*	*Off*
10	*xx*	*N*	*N*	*N*	*Off*	*N*	*N*	*Off*
11	*xx*	*D*	*D*	*vac*	*vac*	*vac*	*vac*	*D*
12	*xx*	*D*	*D*	*E*	*D*	*Off*	*Off*	*Off*
13	*xx*	*D*	*D*	*D*	*E*	*D*	*D*	*D*
14	*xx*	*N*	*N*	*N*	*Off*	*N*	*N*	*N*
15	*xx*	*D*	*D*	*E*	*Off*	*Off*	*Off*	*Off*
16	*xx*	*D*	*D*	*D*	*D*	*S*	*S*	*S*
17	*xx*	*E*	*E*	*D*	*E*	*E*	*Off*	*E*
18	*xx*	*D*	*D*	*E*	*D*	*D*	*Off*	*Off*
19	*xx*	*N*	*D*	*D*	*D*	*D*	*D*	*Off*
20	*xx*	*E*	*D*	*D*	*E*	*D*	*Off*	*Off*
21	*xx*	*Off*	*Off*	*Off*	*D*	*D*	*D*	*Off*
22	*xx*	*N*	*N*	*N*	*N*	*D*	*D*	*D*
23	*xx*	*E*	*E*	*E*	*D*	*D*	*Off*	*Off*
24	*xx*	*E*	*E*	*E*	*Off*	*Off*	*Off*	*D*
25	*xx*	*E*	*E*	*E*	*D*	*D*	*Off*	*E*
26	*xx*	*Off*	*E*	*E*	*E*	*E*	*Off*	*E*
27	*xx*	*Off*	*N*	*N*	*N*	*N*	*D*	*D*
28	*xx*	*Off*	*E*	*E*	*E*	*E*	*Off*	*E*
29	*xx*	*D*	*D*	*D*	*D*	*D*	*Off*	*Off*
30	*xx*	*Off*	*Off*	*Off*	*S*	*Off*	*Off*	*S*

Nurse	Name	D1	D2	D3	D4	D5	D6	D7
31	xx	Off	Off	Off	E	E	D	E
32	xx	Off	Off	D	D	E	E	E
33	xx	Off	Off	Off	S	Off	Off	S
34	xx	Off	Off	Off	E	E	D	E
35	xx	Off	Off	N	N	N	D	D
36	xx	Off	Off	Off	E	E	E	Off
37	xx	Off	Off	D	Off	E	D	Off
38	xx	Off	Off	Off	Off	D	E	Off
39	xx	Off	Off	Off	E	E	E	Off
40	xx	Off	Off	D	D	D	Off	N
41	xx	D	D	D	D	D	Off	Off
42	xx	D	D	D	D	D	Off	Off
43	xx	N	N	N	Off	N	N	N
44	xx	Off	Off	Off	N	Off	Off	Off
45	xx	Off	Off	Off	vac	vac	vac	vac
46	xx	Off	Off	Off	N	N	D	D
47	xx	Off	Off	Off	Off	E	E	Off
48	xx	Off	Off	Off	Off	Off	D	D
49	xx	E1	E1	E1	Off	Off	E	E1
50	xx	Off	Off	Off	N	N	N	Off

Number of fails	:	41
Number of choice points	:	206
Number of variables	:	19967
Number of constraints	:	9673
Running time since creation	:	3.02

Nous pouvons constater que le temps d'exécution est relativement faible et que le nombre d'échecs est peu élevé. Ce problème étant très contraints (en moyenne, la plupart des infirmières ne peuvent travailler que trois ou quatre quarts sur les huits quarts réalisables), de nombreuses réductions de domaines vont avoir lieu lors de la déclaration des premières contraintes (comme les quarts réalisables ou les quarts du lendemain). De cette manière, une grande partie de l'arbre de recherche sera éliminée. De plus, dans le jeu de données, nous pouvons remarquer qu'il y a un certain équilibre entre les infirmières et l'ensemble des quarts réalisables par ces mêmes infirmières. Lors de l'instantiation des variables, cette équilibre doit favoriser la diversité des premiers quarts affectés, et empêcher que des contraintes de quotas soient saturées trop rapidement.
Cependant, cette dernière remarque n'est le fruit que d'une simple observation des données : elle ne constitue pas un argument permettant une bonne stratégie de recherche, et donc, une bonne génération d'horaires. Dans la section suivante, nous alons aborder quelques stratégies pour tenter d'optimiser les temps de calcul et de recherche d'une solution.

5.4 Stratégies à développer

5.4.1 Utilisation de la fonction coût

Plusieurs critères permettent de minimiser la fonction coût afin de trouver la meilleure solution possible. Cependant, nous pouvons nous servir de ces critères pour élaborer une stratégie de recherche amenant vers une bonne

solution. Par exemple, nous pouvons nous servir des préférences (ou aversions) des infirmières pour instantier un ensemble de variables. Si le nombre d'infirmières désirant travailler un certain quart d'un certain jour, est supérieur au maximum d'une contrainte de quota, nous pouvons utiliser d'autres critères pour le choix de variable comme l'ancienneté. Cela permet de propager très rapidement certaines contraintes avant de lancer l'algorithme de recherche, et de réduire la taille de l'arbre de recherche. Cependant, il faut rester prudent sur cette stratégie, puisque rien ne nous certifie que l'instantiation des variables liées aux préférences (ou aversions) peuvent conduire à une solution réalisable (au contraire, elle peuvent engendrer des saturations de contraintes avant même le début de l'algorithme de recherche).

5.4.2 Choix de valeur à partir d'un critère de minimisation

Quand le solveur instantie une variable, il dispose de l'ensemble des valeurs présentes dans le domaine de la variable. Nous voulons développer une heuristique de choix de valeur en quantifiant les contributions relatives à la fonction côut engendrées par l'instantiation de la variable. En effet, si nous pouvons choisir d'instantier une variable avec une valeur qui va produire un coût moins élevé que toute autre valeur du domaine de la variable. D'autres heuristiques du même type peuvent être envisagées : le choix de la valeur peut être en fonction de la propagation des contraintes qu'elle engendre (plus grand nombre de domaines réduits),...,etc.

Conclusion

Le projet consistait à développer un premier prototype permettant de générer des horaires pour le personnel hospitalier, à l'aide de la programmation par contraintes. Si tous les objectifs fixés n'ont pas été atteints, le stage a permis de développer une approche expérimentale avancée de la résolution de ce problème. Certes, les propriétés d'`IlogSolver` pouvant ouvrir de nombreuses perpectives de recherche, beaucoup de temps a été consacré à la discussion de la modélisation. Ensuite, les implantations du module de lecture (développé par Pascal Labit), des contraintes et de la fonction coût m'ont permis de me former au langage C++, et aux techniques de résolution de problème sous contraintes. Sans pour autant être excellents, les premiers résultats fournis par le programme sont encourageants : ils nous permettent de valider les spécificités du programme (respect des contraintes), et de donner des indications pertinentes sur les stratégies de recherche à développer. Enfin, il est important de noter que, même si les modules de la fonction objectif ont été implantés, ils n'ont pas été activés pour la recherche d'une solution. En effet, nous avons privilégié l'optimisation le temps de recherche d'une solution avant de s'occuper à rechercher la meilleure solution. Cependant, ces modules doivent faire l'objet d'une rflexion sur leur utilisation pour développer des algorithmes de recherche permettant d'obtenir des solutions sur l'ensemble de l'horizon, dans des délais convenables.

Même si mon projet ne faisait pas l'objet d'un contrat, il abordait le problème des horaires d'infirmières avec une méthode différente de celle développée par Tsévi Vovor et Pascal Labit (génération de colonnes). Certes, il n'est pas encore possible de comparer les deux méthodes, mais les résultats encourageants obtenus avec la programmation par contraintes laissent envisager la poursuite du projet dans un avenir proche.

Les cinq mois de stage à l'Ecole Polytechnique de Montréal complètent une année de formation dans laquelle il faut inclure les huit mois d'études à l'Université de Sherbrooke. Au dela du savoir que j'aurais acquis, c'est toute l'expérience de la vie et de la découverte du Québec qu'il me faut retenir. De plus, mon autonomie et ma culture personnelle ont été renforcées par mon intégration dans un mode de vie différent et mes rencontres avec des personnes aux cultures diverses.

Chapitre 6

Annexes

6.1 Tableau des Listes

$\mathcal{PA}_n$	Liste des pré-affectations de l'infirmière $n \in calN$.
$\mathcal{FA}_n$	Liste des affectations interdites de l'infirmière $n \in calN$.
$\mathcal{FS}_n$	Liste des quarts réalisables par l'infirmière $n \in calN$.
$\mathcal{W}_n$	Liste des contraintes des charges de travail de l'infirmière $n \in calN$.
$\mathcal{V}_n$	Liste des jours candidats de congé de l'infirmière $n \in calN$.
$\mathcal{Q}$	Liste des contraintes de quota.
$\mathcal{L}_q$	Liste des niveaux d'expertise requis pour la contrainte de quota $q \in \mathcal{Q}$.
$\mathcal{S}_q$	Liste des quarts de la contrainte de quota $q \in \mathcal{Q}$.
$\mathcal{ND}_s$	Liste des quarts du lendemain, pour le quart $s \in \mathcal{S}$.
$\mathcal{WE}$	Liste des fins de semaines de l'horizon.
$\mathcal{Pr}_n$	Liste des préférences (et aversions) individuelles de l'infirmière $n \in calN$.

6.2 Extrait d'un fichier de données

General_Info

ProblemID_Lot_number	6230
ProblemID_Task_number	1
ProblemSize_No_of_nurses	5
ProblemSize_Starting_date	2000/08/20
ProblemSize_Ending_date	2000/09/30
ProblemSize_No_of_skill_levels	12
ProblemSize_No_of_daily_quota_periods	5
ProblemSize_No_of_possible_shifts	8

Pref(+)_and_aver(-)

OptimizerControl_Sal_weight	0
OptimizerControl_Pref_weight	100

Aversion

Aversion_For_Overstaffing	−50
Aversion_For_Understaffing	−50
Aversion_For_no_long_weekend	−50

GroupPreferences.Other_group_requests

Skill_Levels	*Shifts*	*Quota_Periods*	*Days*	*Preference(0_10)_or_Aversion(−10_0)*
nil	*nil*	*nil*	*nil*	*nil*

Optimizer_control

Time_limit(hours)	2
Max_no_of_schedules	5
Stop_at_1st_feasible_sch	1
Max_integrality_gap	0

MasterProbC_L_relax_plateau

Iteration_limit_before_br	1
Min_percent_improvement	1
Max_local_branching_depth	1

Quotas.Quota_periods

Period_number	*Start*	*End*
1	7 : 30	11 : 30
2	11 : 30	15 : 30
3	15 : 30	19 : 30
4	19 : 30	23 : 30
5	23 : 30	7 : 30

ProblemSize.Shift_List

Symbol	*Hours_Paid*	*Type*(d,e,n)	*Periods*	*Min_Off*	*Max_Off*	*Next_Day_Shifts*
D	8 : 00	d	1, 2	1	20	$D, D12, E, E1, E2, S, S12$
$D12$	12 : 00	d	1, 2, 3	2	15	$D, D12, E, E1, E2, S, S12$
E	8 : 00	e	3, 4	2	15	$D, D12, E, E1, E2, S, S12$
N	8 : 00	n	5	2	15	$D, D12, E, E1, E2, N, S, S12$
$S12$	12 : 00	n	4, 5	2	15	$S12, S$
$E1$	4 : 00	e	3	1	20	$D, E1$
$E2$	4 : 00	e	4	1	20	$S, S12, E, E1, E2$
S	8 : 00	n	5	2	15	$S12, S$

Quotas_quota_Constraints

Skill_Levels	*Shifts*	*Sense*	*Quotas*	*Deficit*	*Surplus*	*Period*	*Days*
5, 7, 8, 9, 10, 11, 12	$D, D12$	=	9	2	2	1	Mo, Tu, We, Th, Fr
5, 7, 8, 9, 10, 11, 12	$D, D12$	=	9	2	2	2	Mo, Tu, We, Th, Fr
5, 7, 8, 9, 10, 11, 12	$D12, E, E1$	=	8	2	2	3	Mo, Tu, We, Th, Fr
5, 7, 8, 9, 10, 11, 12	$E, E2, S12$	=	8	2	2	4	Mo, Tu, We, Th, Fr
5, 7, 8, 9, 10, 11, 12	$N, S12, S$	=	7	2	2	5	Mo, Tu, We, Th, Fr
5, 7, 8, 9, 10, 11, 12	$D, D12$	=	7	2	2	1	Sa, Su
5, 7, 8, 9, 10, 11, 12	$D, D12$	=	7	2	2	2	Sa, Su
5, 7, 8, 9, 10, 11, 12	$D12, E, E1$	=	7	2	2	3	Sa, Su
5, 7, 8, 9, 10, 11, 12	$E, E2, S12$	=	7	2	2	4	Sa, Su
5, 7, 8, 9, 10, 11, 12	$N, S12, S$	=	7	2	2	5	Sa, Su

Nurse_List (1 nurse skills from 50)

NurseID.Identification

Employee_Number	977023
FirstName_LastName	*Monica_Hannan*
Skill_Level	8
Seniority(*non_null_value*)	1

Current_Sch_Info

HardCon_Feasible_Shifts	$D, D12, S, S12$
SoftCon_Max_Assgts	30
SoftCon_Min_Assgts	30

HardConstraints_Active

Off_Weekends	1
Hol_and_Vac_to_Grant	0

HardConstraints_Hol_Vac

Earliest_Day	*nil*
Latest_Day	*nil*
Min_number_of_days_off	0
Max_number_of_days_off	0
Candidate_days	*nil*

Int_Constraints_Shift

Max_day_shift_ratio	50
Min_day_shift_ratio	50
Max_evening_shift_ratio	50
Min_evening_shift_ratio	50
Max_night_shift_ratio	50
Min_night_shift_ratio	50

HardCons_Consecutive_assign

Max_cons_days_off	3
Min_cons_days_off	2
Max_cons_day_shifts	2
Min_cons_day_shifts	1
Max_cons_eve_shifts	2
Min_cons_eve_shifts	1
Max_cons_night_shifts	2
Min_cons_night_shifts	1
Max_cons_week_ends_on	1
Min_cons_week_ends_on	1
Max_cons_week_ends_off	1
Min_cons_week_ends_off	1

HardConstraints.Workload

Starting_date	*End_date*	*Min_hours_paid*	*Max_hours_paid*
2000/08/20	2000/09/02	80	80
2000/09/03	2000/09/16	80	80
2000/09/17	2000/09/30	80	80

SetOfAssignmentValues.Salary

Value(x10)	*Days*	*Shifts*
nil	*nil*	*nil*

SetOfAssignValues.Pref_or_Aver

Value	*Days*	*Shifts*
nil	*nil*	*nil*

HardConstraints.Pre-assignments

Days	*Shifts*
nil	*nil*

HardConstraints.Forbidden_assign

Days	*Shifts*
nil	*nil*

SetOfAssignments.Previous_Sch_Info

Duty_Days	*Shifts*
nil	*nil*

Hol_+_Vac_Granted_prev_Horizon nil

SetOfAssign.Init_Resource_of_Prev_Sch

Weekend	*Rotation*
0	0

Assign_Preceding_First_One_of_Prev_Sch

Duty_Day	*Shift*
nil	*nil*

Bibliographie

[1] Gilles Pesant.
Scheduling Physicians by Constraint Programming.
Article développant la génération d'horaires de médecins par la programmation par contraintes.

[2] Brigitte Jaumard, Fréderic Semet, Tsévi Vovor.
A generalized linear programming model for nurse scheduling.
Article développant la génération d'horaires d'infirmières par la programmation linéaire.

[3] Brigitte Jaumard, Pascal Labit, Tsévi Vovor.
Description des contraintes et des objectifs du programme IRIS de confection automatique d'horaires d'infirmières.
Document de référence pour la réalisation du fichier de données à l'intention de l'infirmière chef.

[4] Tsévi Vovor.
Problèmes de chemins bicritères ou avec contraintes de ressourses :algorithmes et applications.
PH.D.Thesis-Ecole Polytechnique de Montréal.

[5] Pascal Labit.
Génération autommatisée d'horaires pour les infirmières.
Rapport de Maitrise-Ecole Polytechnique de Montréal, août 2000.

[6] Philippe Galinier.
Horaires d'infirmières.
Présentation du problème des horaires d'infirmières et du fichier de données.

[7] Kim Mariott, Peter J. Stuckey.
Programming with Constraints, An Introduction.
1998, Massachussetts Institute of Technology.
Manuel d'introduction à la programmation par contraintes.

Table des matières

1 Le RCM.$_2$, le GERAD et le CRT **3**
- 1.1 Le R.C.M.$_2$ . . . 3
- 1.2 le GERAD . . . 3
- 1.3 Le C.R.T. . . . 4
- 1.4 La programmation scientifique du C.R.T. . . . 5
  - 1.4.1 Les objectifs . . . 5
  - 1.4.2 Axes et programmes de recherche . . . 5

2 Introduction au problème **7**
- 2.1 Introduction . . . 7
- 2.2 Description du problème . . . 7
- 2.3 Présentation des contraintes . . . 8
  - 2.3.1 Les contraintes dures (hard constraints) . . . 9
  - 2.3.2 La fonction objectif (cost function) . . . 14
  - 2.3.3 La fonction objectif . . . 17

3 La programmation par contraintes **18**
- 3.1 La représentation du problème . . . 18
- 3.2 La recherche de solution . . . 19
  - 3.2.1 L'action des contraintes . . . 20
  - 3.2.2 L'échec, et le retour arrière (backtracking) . . . 21

4 Implantation du problème **22**
- 4.1 La lecture et la modélisation des données . . . 22
- 4.2 Choix de la modélisation . . . 22
  - 4.2.1 Le modèle centré sur les infirmières . . . 22
  - 4.2.2 Le modèle centré sur les quarts . . . 23
- 4.3 L'implantation des contraintes . . . 24
  - 4.3.1 Les contraintes dures . . . 24

5 Stratégies et résultats **32**
- 5.1 Heuristique de choix de variable . . . 32
  - 5.1.1 L'ordre lexicographique . . . 32
  - 5.1.2 Choix du plut petit domaine . . . 33

5.2 Heuristique de choix de valeur 33
5.2.1 L'ordre lexicographique 33
5.2.2 Le choix aléatoire 33
5.3 Test expérimentaux 33
5.3.1 L'ordre lexicographique 33
5.3.2 Stratégie de choix aléatoire 36
5.3.3 Test sur le fichier des 50 infirmières 37
5.4 Stratégies à développer 38
5.4.1 Utilisation de la fonction coût 38
5.4.2 Choix de valeur à partir d'un critère de minimisation 39

6 Annexes **41**
6.1 Tableau des Listes 41
6.2 Extrait d'un fichier de données 42